Алексеев В.В.

ИНФОРМАТИКА

(учебно-методическое пособие)



СарФТИ НИЯУ МИФИ

**Саровский физико-технический институт-
филиал Национального исследовательского
ядерного университета «МИФИ»**

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ЭЛЕКТРОНИКИ
Кафедра вычислительной и информационной техники

Алексеев В.В.

ИНФОРМАТИКА

Учебно-методическое пособие
(Конспект лекций)

Саров
2022

УДК 621.
ББК 22.176
А47

Алексеев В.В. – ИНФОРМАТИКА.
Учебно-методическое пособие. СарФТИ НИЯУ МИФИ, 2022 г.

Данное пособие представляет собой конспект лекций по курсу «Информатика» и предназначено для студентов младших курсов бакалавриата, обучающихся по направлениям подготовки «09.00.00-Информатика и вычислительная техника».

В пособии изложены основы современной информатики как комплексной научно-технической дисциплины, включающей изучение структуры и общих свойств информации и информационных процессов, общих принципов построения вычислительных устройств, а также систем обработки, хранения и передачи информации.

Теоретические сведения иллюстрируются примерами, рисунками, таблицами.

Учебно-методическое пособие может быть также полезно студентам и других направлений подготовки, использующих отдельные разделы информатики в направлениях своей подготовки.

© Издательство СарФТИ 2022

Содержание

Наименование раздела	Стр
Лекция 1. Введение.	3
1.1Основные понятия	3
1.2 Основные этапы развития ВТ	6
Лекция 2 Общие сведения об ЭВМ. Принцип действия ЭВМ	12
2.1 Принцип действия ЭВМ	12
2.1.1 Принцип программного управления	14
2.2 Структурная организация ЭВМ	21
2.2.1 Модульный принцип построения ЭВМ	22
2.2.2 Варианты структур ЭВМ	23
2.3 Технико-эксплуатационные характеристики ЭВМ	29
2.4 Классификация ЭВМ	31
Лекция 3. Основы теории информации	39
3.1 Информация	39
3.1.1 Структурная мера информации	41
3.1.2 Статистическая мера информации	42
3.1.3 Основные свойства энтропии	46
3.2 Количество информации	47
3.3 Основные информационные характеристики	48
Лекция 4. Представление числовой информации в цифровых устройствах	50
4.1 Позиционные системы счисления	50
4.2 Перевод числовой информации из одной ПСС в другую	54
4.2.1 Перевод целых чисел делением на основание новой ПСС	54
4.2.2 Перевод правильных дробей умножением на основание новой ПСС	55
4.2.3 Табличный метод перевода	57
4.2.4 Использование промежуточной системы счисления	59
4.3 Формы представления чисел	59
4.3.1 Представление чисел в формате с фиксированной точкой	60
4.3.2 Представление чисел в формате с плавающей точкой	61
4.3.3 Представление отрицательных чисел	64
4.3.4 Погрешности представления чисел	67
Лекция 5. Основы двоичной арифметики	73
5.1 Формальные правила двоичной арифметики	73
5.2 Сложение чисел в формате с фиксированной точкой на двоичных сумматорах	74
5.2.1 Двоичный сумматор прямого кода (ДСПК)	74
5.2.2 Двоичный сумматор дополнительного кода (ДСДК)	75
5.2.3 Двоичный сумматор обратного кода (ДСОК)	77
5.3 Переполнение разрядной сетки	80
5.4 Сложение чисел в формате с плавающей точкой	82
5.5 Операция нормализации	84

5.6 Оценка точности выполнения арифметических операций 88
5.6.1 Погрешности выполнения арифметических операций 89
5.6.2 Погрешности округления 89
Лекция 6 Кодирование информации 92
6.1 Кодирование текстовой информации 93
6.2 Кодировки UNICODE 95
6.2.1 Кодовое пространство 96
6.2.2 Правила записи одного символа в UTF 99
6.3 Представление графической информации 103
6.3.1 Растровая графика 104
6.3.2 Векторная графика 106
6.3.3 Фрактальная графика 110
6.3.4 Цветовые модели компьютерной графики 115
6.4 Представление звуковой информации 122
6.4.1 Форматы звуковых файлов 126
Лекция 7 Логические основы ЭВМ. 131
7.1 Основные определения 131
7.2 Элементарные функции алгебры логики 133
7.3 Выражение одних элементарных функций через другие 136
7.4 Законы алгебры логики 138
7.5 Свойства конъюнкции, дизъюнкции 139
7.6 Представление функций алгебры логики 140
7.7 Логические элементы и их применение 142
7.8 Функционально полные системы функций алгебры логики 145
Лекция 8 Процессор Архитектура ЭВМ 151
8.1 Структура процессора 151
8.1.1 Последовательность выполнения команд процессора 152
8.2 Процессорная архитектура 153
8.2.1 Производительность 154
8.2.2 CISC-архитектура 155
8.2.3 Конвейерная обработка команд 156
8.2.4 Суперскалярная обработка 158
8.2.5 RISC-архитектура 159
8.2.6 VLIW-архитектура 162
8.3 Структура команд и адресация 166
8.4 Система прерываний 175
Лекция 9 Шины
9.1 Шина PCI Express 181
9.2 Прямой доступ к памяти 181
Лекция 10 Память ЭВМ 185
10.1Классификация памяти 186
10.1.1 Полупроводниковая память 186
10.1.2 Электромеханическая память 187
10.2 Многоуровневая память 191
Лекция 11 Внешние устройства 193

11.1 Устройства ввода 193
11.1.1Клавиатура 193
11.1.2 Компьютерные мыши 195
11.1.3 Трекбол и джойстик 196
11.1.4 Сенсорная панель 197
11.1.5 Сканеры 200
11.2 Устройства вывода 201
11.2.1 Мониторы и дисплеи 201
11.2.2Принтеры 204



Лекция 1.

1.Введение

1.1 Основные понятия

Информа́тика (фр. Informatique; англ. Computer science) - наука о методах и процессах сбора, хранения, обработки, передачи, анализа и оценки ***информации*** с применением компьютерных технологий, обеспечивающих возможность её использования для принятия решений.

Информатика включает дисциплины, относящиеся к обработке информации ***в вычислительных машинах*** и ***вычислительных сетях***: как абстрактные, вроде анализа ***алгоритмов,*** так и конкретные, например, разработка ***языков программирования*** и ***протоколов передачи данных*** и другое.

Темами исследований в информатике являются вопросы: что можно, а что нельзя реализовать в программах и базах данных (***теория вычислимости и искусственный интеллект***), каким образом можно решать специфические вычислительные и информационные задачи с максимальной эффективностью (***теория сложности вычислений***), в каком виде следует хранить и восстанавливать информацию специфического вида (***структуры и базы данных***), как программы и люди должны взаимодействовать друг с другом (***пользовательский интерфейс и языки программирования, представление знаний***) и т. п.

На основании вышеизложенного можно привести определения понятия информатика, как, например:

- **Информатика** – научная дисциплина, изучающая структуру и общие свойства информации, а также вопросы, связанные с её сбором, хранением, поиском, переработкой, преобразованием, распространением и использованием в различных сферах деятельности;

Приведенные определения носят общий характер и не позволяют провести чёткую грань между информатикой и другими научными дисциплинами, например, математикой и биологией. В данном курсе информатика будет рассматриваться в более узком смысле, применительно к цифровой вычислительной технике и здесь можно ограничиться, например, таким определением:

- **Информатика** – научная дисциплина, изучающая законы и методы накопления, передачи и обработки информации с помощью ЭВМ.

Тогда в соответствии с этим определением можно привести основные направления информатики как научной дисциплины и которыми являются:

- разработка вычислительных систем и программного обеспечения;

- теория информации, изучающая процессы, основанные на передаче, приеме, преобразовании и хранении информации;
- методы, позволяющие создавать программы для решения задач, требующих определенных интеллектуальных усилий при использовании их человеком (логический вывод, понимание речи, визуальное восприятие и др.);
- системный анализ, состоящий в изучении назначения проектируемой системы и в определении требований, которым она должна соответствовать;
- методы анимации, машинной графики, средства мультимедиа;
- телекоммуникационные средства (глобальные компьютерные сети);
- различные приложения, которые используются в производстве, науке, образовании, медицине, торговле, сельском хозяйстве и др.

Конечно, в современном мире роль информатики огромна. Она охватывает не только сферу материального производства, но и интеллектуальную, духовную стороны жизни. Увеличение объемов производства компьютерной техники, развитие информационных сетей, появление новых информационных технологий значительно влияют на все сферы общества: производство, науку, образование, медицину, культуру и т. д.

В США информатика в таком понимании получила название ***Computer Science***.

В основе информатики лежат два фундаментальных понятия: ***информация*** и ***алгоритм***. Вместе с основными философскими категориями материя, энергия, время они образуют основу, на которой строится вся система человеческих знаний. Но дать точные определения этим понятиям нельзя. Вполне очевидно, что их определения будут приблизительны, и поэтому их можно вполне назвать интуитивными. Вот одно из этих определений:

- **Информация** – это любые сведения, об объектах и явлениях окружающего нас мира, их свойствах, характеристиках и состоянии, воспринимаемые информационными системами.

Сразу следует отметить, что информация является характеристикой не сообщения, а соотношения между сообщением и его анализатором. Если отсутствует потребитель, хотя бы потенциальный, говорить об информации не имеет смысла.

В информатике под информацией понимают некоторую последовательность символических обозначений (букв, цифр, слов, образов, звуков и т. п.), которые несут смысловую нагрузку и

представлены в виде, обеспечивающем их обработку средствами вычислительной техники.

- **Алгоритм** - всякое точное предписание, которое задает вычислительный (или алгоритмический) процесс, начинающийся с произвольно допустимых для данного процесса исходных данных и направленный на получение полностью определяемого этими исходными данными результата.

Более детальное определение алгоритма и его свойства будут рассмотрены ниже.

Все ЭВМ делятся на аналоговые и цифровые. В данном курсе рассматриваются только цифровые ЭВМ (компьютеры), и поэтому в дальнейшем под ЭВМ будут пониматься только цифровые ЭВМ, т.е. слово «цифровые» можно опустить. Более детальная классификация ЭВМ будет рассмотрена ниже.

В ЭВМ алгоритм представляется в виде последовательности команд (логических, арифметических и т. д.) называемой программой.

Каждая команда выполняется аппаратно. В специализированных ЭВМ бывают команды, по сложности выполняемых преобразований информации не уступающие небольшой программе, написанной в командах для универсальной ЭВМ. Теоретически можно построить ЭВМ, выполняющую некоторый заданный алгоритм аппаратно без хранимой в памяти программы. Практическая ценность такой ЭВМ будет невелика, так как она будет очень дорога и иметь крайне узкую область применения.

Примечание. Между аппаратным и программным уровнями в некоторых ЭВМ существует ещё один уровень, называемый микропрограммным. Академик В. М. Глушков, используя микропрограммный способ управления, встроил в ЭВМ МИР-2 язык высокого уровня Аналитик. Таким образом, ЭВМ МИР-2 не имела традиционной системы команд (машинных кодов). Её командами были операторы языка высокого уровня.

Для того чтобы создавать эффективные ЭВМ и качественное программное обеспечение, необходимо знать логические основы построения ЭВМ и методы создания программ. Построение логических схем (материальных объектов), из которых состоит ЭВМ, очень близко к электронике и автоматике (техническим дисциплинам), а написание программ близко к математике (теории алгоритмов, численным методам и т. д.), т. е. к нематериальной категории. Информатика (computer science) объединяет все необходимые знания из областей построения схем и

написания программ в единую дисциплину, овладение которой позволяет решать сложнейшие задачи проектирования и использования ЭВМ.

1.2 Основные этапы развития ВТ

Появление ЭВМ было подготовлено историческим развитием средств вычислений (Абак, счеты и т.п.). Первое механическое счетное устройство (суммирующее), разработанное Б. Паскалем, появилось в 1624г. В России первое суммирующее устройство было изобретено и изготовлено в 1770г. Е. Якобсоном, часовых дел мастером в г. Несвиже. Суммирующие устройства того времени не имели применения: сложны, ненадежны. Коренной перелом в создании счетных устройств произошел в середине XIX в., когда появилась необходимая технологическая база. В середине 80-х годов XIX в. удалось организовать промышленный выпуск суммирующих устройств. Параллельно с развитием счетных устройств создавались и арифмометры. Первым в мире арифмометром стала арифметическая машина Г. Лейбница, появившаяся в конце XVII в. Машина не получила широкого распространения, но основная идея Лейбница - идея ступенчатого вала - осталась действенной и плодотворной даже и XX в. На принципе ступенчатого вала был построен и арифмометр Томсона - первое в мире счетное устройство, изготовляемое промышленно. Изготавливались и другие арифмометры, но основным элементом было зубчатое колесо Однера с переменным числом зубьев.

В 1878 г. в России П.Л. Чебышевым был создан арифмометр, преимуществом которого являлось то, что перенос десятков из младшего разряда в старший происходил постепенно в процессе накопления единиц и распространялся на последующие разряды. Идеи, заложенные в арифмометре Чебышева, лежат в основе многих видов современных вычислительных устройств.

Существенным недостатком суммирующих устройств и арифмометров считается невозможность значительного увеличения скорости вычислений. Впервые был автоматизирован процесс вычислений английским ученым Ч. Бебиджем, при этом скорость вычислений определялась значениями: сложение 1 с., умножение 1 мин. Идеи Ч. Бебиджа не были поддержаны современниками, и к ним обратились только в 40-х годах XX столетия.

В 1947г. была создана релейная вычислительная машина “Марк-2”, в которой впервые использовалась двоичная система счисления, а в качестве запоминающих устройств использовались реле (более 13 тыс. штук). Производительность машины определялась параметрами: сложение

0.125 с., умножение 0.25 с. (Наиболее удачной из релейных вычислительных машин была релейная вычислительная машина, сконструированная советским инженером Н.И. Беловым в 1956г.)

В 1943г. в Гарвардском университете под руководством американских ученых Д. Моучли и Д. Эккерта приступили к созданию электронной вычислительной машины (ЭВМ). Машина изготавливалась по заказу артиллерийского управления МО США и была предназначена для расчета баллистических таблиц. Завершенная в 1945 г. и получившая название ЭНИАК, эта ЭВМ имела громадные размеры, содержала 18 тыс. ламп, более 15 тыс. реле, потребляла 150 кВт. Скорость вычислений определялась параметрами: сложение 0.2 мс., умножение 2.8 мс.

Несоответствие между временем решения и временем подготовки задачи к решению вручную было настолько большим, что выигрыш от скорости вычисления почти полностью покрывал время на подготовку. Создание ЭНИАК - начало бурного развития ЭВМ 1-го поколения. Существенно упростилась процедура решения последовательности специальных задач в модели ЭНИАК (1948г.), впервые управляемой от памяти (постоянного типа), но автоматический ввод новых программ и автоматическое их ``самоизменение`` с построением разветвлений еще не были доступны.

Следующий скачок - 50-е годы. В СССР первая малая ЭВМ (МЭСМ) была создана под руководством С.Л. Лебедева в 1951г., где были воплощены новые технические принципы.

Следует отметить, что МЭСМ была введена в действие на два года раньше, чем Дж. Нейман, с именем которого связывают за рубежом разработку принципов универсальных ЭВМ, создал свою ЭДВАК. Более того, МЭСМ относилась к машинам с последовательно-параллельной обработкой кодов, и может рассматриваться как базовый прототип быстродействующих ЭВМ, чего нельзя сказать об американских ее ровесниках - машинах с последовательной обработкой кодов.

На рис.1.1 приведена схема перехода от режима ручного программирования к автоматическому управлению.[1]

В процессе создания МЭСМ разрабатывались, монтировались и опробовались быстродействующие узлы и устройства большой электронно-счетной машины (БЭСМ), монтаж и отладка которой были завершены в 1953г. В течение нескольких последующих лет БЭСМ, производившая 8 тыс.оп./сек. была самой быстро-действующей машиной в Европе. Структура и основные схемы БЭСМ стали классическими: они

[1] Л.Г. Хоменко. К истории развития отечественной вычислительной техники. УсиМ, №6,1987г.

были положены в основу БЭСМ-2, М2 и др. В 1953г. под руководством Ю.А. Базилевского была создана ЦВМ “Стрела”, а в 1954г. и под руководством Б.И. Рамеева - ЭВМ “Урал”. Почти одновременно с этими машинами появились такие ЭВМ, как М-3, Минск-1 и др.



Рис.1.1

Существует условное разделение развитие средств вычислительной техники на поколения, которые сменялись постепенно, и поэтому временные границы между поколениями размыты (Таблица 1.1).

Поколение ЭВМ определяется совокупностью взаимосвязанных и взаимообусловленных существенных особенностей и характеристик, используемых при построении машин, конструктивно-технологической (в первую очередь элементной) базы и реализуемой в машине архитектуры (логической организации).

Первое поколение образовывали ламповые ЭВМ, промышленный выпуск которых начался в начале 50-х годов. В качестве компонентов логических элементов использовались электронные лампы. ЭВМ I^{го} поколения характеризуются наличием параллельного арифметического устройства, разделением памяти на оперативную и внешнюю, применением перфокарт и перфолент как носителей информации при вводе и выводе данных. Среднее быстродействие ~10 тыс. оп./сек.

Основу технической базы ЭВМ II^{го} поколения, появившихся в конце 50-х годов, составляли полупроводниковые приборы. ЭВМ II^{го} поколения характеризуются повышением надежности, снижением потребляемой мощности, уменьшением габаритов. Характерен параллелизм в работе отдельных блоков. К ЭВМ этого поколения относят:

- малые ЭВМ серий “Наири”, “Мир”;

- средние ЭВМ (5-30 тыс. оп./сек.): Минск-2, Минск-22, Минск-32, Урал-14, Раздан-3, БЭСМ-4, М220;
- управляющие вычислительные машины: Днепр, ВНИИЭМ и др.

Поколения ЭВМ Таблица 1.1

Поколение	Элементная база процессора	Макс. емкость ОЗУ, байт	Макс. быстро-действие процес-сора, оп/с	Основные языки програм-мирования	Управление ЭВМ пользователем
Первое 1951-1954	электронные лампы	10^2	10^4	Машинный код	Пульт управления и перфокарты
Второе 1958-1960	транзисторы	10^3	10^6	Ассемблер	Перфокарты и перфоленты
Третье 1965-1968	ИС	10^4	10^7	Процедур-ные языки высокого уровня (ЯВУ)	Алфавитно-цифровой терминал
Четвертое 1976-1979	БИС	10^5	10^8	Процедур-ные ЯВУ	Монохромный или графический дисплей, клавиатура
Четвертое с 1985	СБИС	10^7	10^9	Процедур-ные ЯВУ	Цветной графический дисплей, клавиатура, «мышь» и др.
Пятое	Усовершен-ствованные СБИС	10^8	10^{12}	Языки логического программи-рования	Цветной графический дисплей и устройства голосовой связи

Наиболее мощной отечественной ЭВМ II^{го} поколения являлась БЭСМ-6, созданная под руководством академиков С.А. Лебедева и В.А. Меньшова, с производительностью 1 млн. оп./сек.

Второе поколение позволило существенно расширить сферу использования вычислительной техники, приступить к созданию АСУ отраслями, предприятиями, технологическими процессами.

Дальнейшее увеличение быстродействия ЭВМ тормозилось конструктивным выполнением электронных схем, собираемых из отдельных элементов. Стремление к повышению быстродействия, надежности, снижению стоимости аппаратуры привело к появлению новой

элементной базы ВТ - интегральных микросхем, составляющих элементную базу ЭВМ третьего поколения.

ЭВМ III[го] поколения появились во второй половине 60-х годов, когда фирма IBM (США) разработала систему машин IBM-360. Эта система оказала влияние на логическую организацию машин общего назначения, разработанных в других странах. ЭВМ III[го] поколения оперируют с произвольной буквенно-цифровой информацией. В них фактически соединились два направления предыдущих поколений машин. ЭВМ для делового, коммерческого применения с обработкой алфавитной информации и ЭВМ, предназначенных для научных учреждений и обработки цифровой информации. ЭВМ III[го] поколения построены по принципу независимой параллельной работы различных устройств: процессоров, средств внешней памяти и т.д. Типичные представители ЭВМ III[го] - машины единой серии ЕС ЭВМ, предусмотренные для решения научно-технических, экономических и управленческих задач в разного рода АСУ и системах обработки данных. Промышленный выпуск ЕС ЭВМ начат в 1972г. (СССР, Болгария, Венгрия, Германия (ГДР), Польша, Чехословакия).

Конструктивно-технологической основой ЭВМ IV поколения, контуры которого определить довольно трудно, являлись БИС и СБИС. В первую очередь следует отметить построение блоков памяти с применением БИС. К IV поколению относятся реализованные на СБИС такие средства ВТ как микропроцессоры и микро-ЭВМ, микроконтроллеры. В рамках ЭВМ IV[го] поколения созданы новые по ряду эксплуатационных характеристик и их способу использования типы вычислительных устройств - персональные компьютеры. Характерным для крупных ЭВМ IV[го] поколения является наличие нескольких процессоров, ориентированных на выполнение определенных операций, процедур или на решение некоторых классов задач. В рамках этого поколения создаются многопроцессорные вычислительные системы с быстродействием несколько миллионов оп./сек. и многопроцессорные управляющие комплексы повышенной надежности. Примерами крупных вычислительных систем, которые следует отнести к четвертому поколению, являлись многопроцессорные комплексы "Эльбрус-2" и "Эльбрус-3" с суммарным быстродействием соответственно 100 млн. и порядка 1 млрд. операций/сек.; многопроцессорная вычислительная система ПС-2000, содержащая до 64 процессоров, управляемых общим потоком команд, в которой при распараллеливании процесса выполнения программ может быть достигнуто быстродействие до 200 млн. оп./сек.

В контурах нового, пятого поколения ЭВМ (а также вычислительных систем) они должны при более высокой производительности, надежности и более низкой стоимости обладать качественно новыми свойствами:

- возможностью взаимодействия с ЭВМ при помощи естественного языка, человеческой речи, графических изображений;
- способностью системы самообучаться, производить ассоциативную обработку информации, делать логические суждения, вести диалог в форме вопросов и ответов, принимать решения, адекватные имеющейся в ЭВМ информации;
- способностью системы "понимать" содержимое базы данных и использовать ее при решении различных задач;
- и т.п.

Элементная база средств вычислительной техники – СБИС с использованием ***опто-*** и ***криоэлектроники***.

Оптоэлектроника – раздел электроники, связанный с эффектами взаимодействия оптического излучения с электронами в веществах (главным образом в твердых телах) и использованием этих эффектов для генерации, передачи, хранения, обработки и отображения информации.

Криоэлектроника (криогенная электроника) – область науки и техники, занимающаяся применением явлений, имеющих место в твердых телах при температуре ниже 120 К (криогенных температурах) в присутствии электрических, магнитных или электромагнитных полей (явление сверхпроводимости), для создания электронных приборов и устройств.

Лекция 2.

2 Общие сведения об ЭВМ

2.1 Принцип действия ЭВМ

ЭВМ - техническая система, предназначенная для выполнения вычислений на основе алгоритмов. Поскольку точного и однозначного определения алгоритма не существует, приведем еще раз одно из наиболее распространенных его определений.

Алгоритм - всякое точное предписание, которое задает вычислительный (или алгоритмический) процесс, начинающийся с произвольно допустимых для данного процесса исходных данных и направленный на получение полностью определяемого этими исходными данными результата.

(Алгоритм - центральное понятие кибернетики, происходящее от латинского написания имени узбекского математика IX века аль Хорезми, родившегося в Хиве-Algoritmi)

Алгоритм характеризуется:

- совокупностью возможных исходных данных;
- совокупностью возможных конечных результатов;
- совокупностью промежуточных результатов;
- правилом начала;
- правилами переработки;
- правилом окончания;
- правилом извлечения результата.

Из приведенного определения понятия алгоритма следуют его особенности, которые состоят в следующем:

- ***Конечность.*** Алгоритм всегда должен заканчиваться после выполнения конечного числа шагов;
- ***Определенность***. Каждый шаг алгоритма должен быть определен;
- ***Ввод***. Алгоритм имеет некоторое число входных данных, задаваемых до начала его работы;
- ***Вывод.*** Алгоритм имеет одно или несколько выходных данных;
- ***Эффективность.*** Алгоритм считается эффективным, если все его операции можно точно выполнить в течение конечного промежутка времени.

Естественно, что для алгоритма характерны три вида данных:

- ***входные*** (исходные) данные;
- ***промежуточные*** данные;
- ***выходные*** данные.

В общем случае, используемые данные алгоритмом и их взаимодействие можно представить в виде условной функциональной схемы, представленной на рис. 2.1.

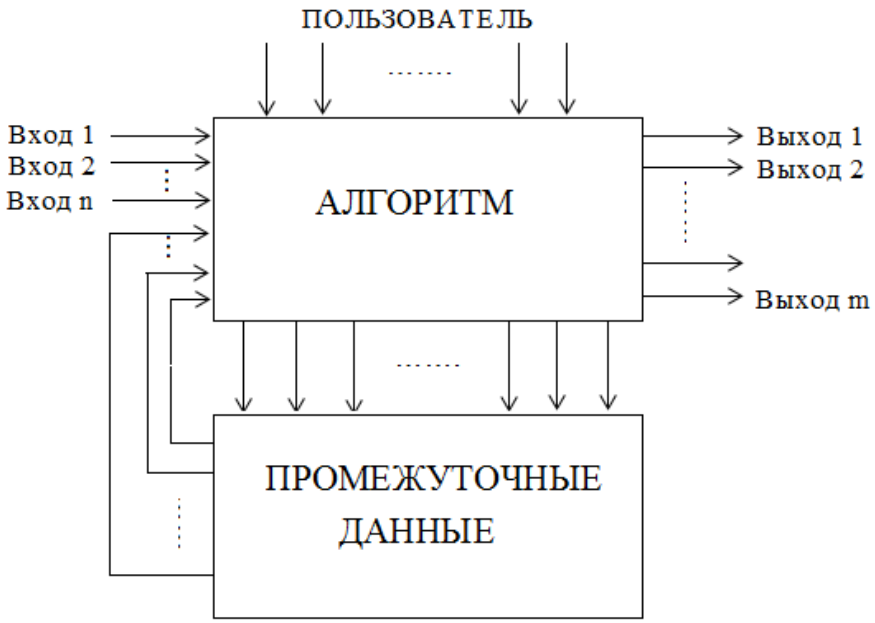


Рис. 2.1

Важно отметить, что одной из главных особенностей алгоритма является то, что на одинаковый набор исходных данных всегда выдается одинаковый набор выходных данных.

В соответствии с вышеизложенным вполне стали определимы характерные черты алгоритма, четко вырисовывающиеся из его интуитивного определения и признающихся характерными для его понятия:

1. Алгоритм – это процесс последовательного построения величин, идущий в дискретном времени таким образом, что в начальный момент задается исходная конечная система величин (***правило начала***), а в каждый следующий момент получается новая система по определенному закону (программе) из системы величин, имевшихся в предыдущий момент времени (***дискретность алгоритма***);
2. Система величин, получаемых в какой-то (не начальный) момент времени, однозначно определяется системой величин, полученных в предшествующие моменты времени (***детерминированность алгоритма***);
3. Закон получения последующей системы величин из предшествующей должен быть простым и локальным (***элементарность шагов алгоритма***). (Типичный пример множества элементарных действий – система команд ЭВМ);

4. Если способ последующей величины из какой-нибудь заданной величины не дает результата, то должно быть указано, что надо считать результатом алгоритма (***результативность алгоритма***);
5. Начальная система величин может выбираться из некоторого потенциально бесконечного множества (***массовость алгоритма***)

Вполне очевидно, что понятие алгоритма, определяемое этими требованиями, конечно, не строгое, так как в формулировках встречаются слова «способ», «величина», «простой», «локальный» и др., точный смысл которых не установлен, и поэтому это нестрогое понятие алгоритма называют ***непосредственным***, или ***интуитивным понятием алгоритма***.

2.1.1 Принцип программного управления

ЭВМ строятся на основе ***принципа программного управления***, реализация которого была впервые предложена в 1946г. ***Дж. фон Нейманом*** и используется в настоящее время, Этот принцип известен как ***Архитектура фон Неймана*** (модель фон-Неймана, Принстонская архитектура (рис.2.2). Главное в этой модели то, что используется принцип совместного хранения команд и данных в памяти компьютера. Вычислительные машины такого рода часто обозначают термином «***машина фон Неймана***», однако соответствие этих понятий не всегда однозначно. В общем случае, когда говорят об архитектуре фон Неймана, подразумевают принцип хранения данных и инструкций в одной памяти.

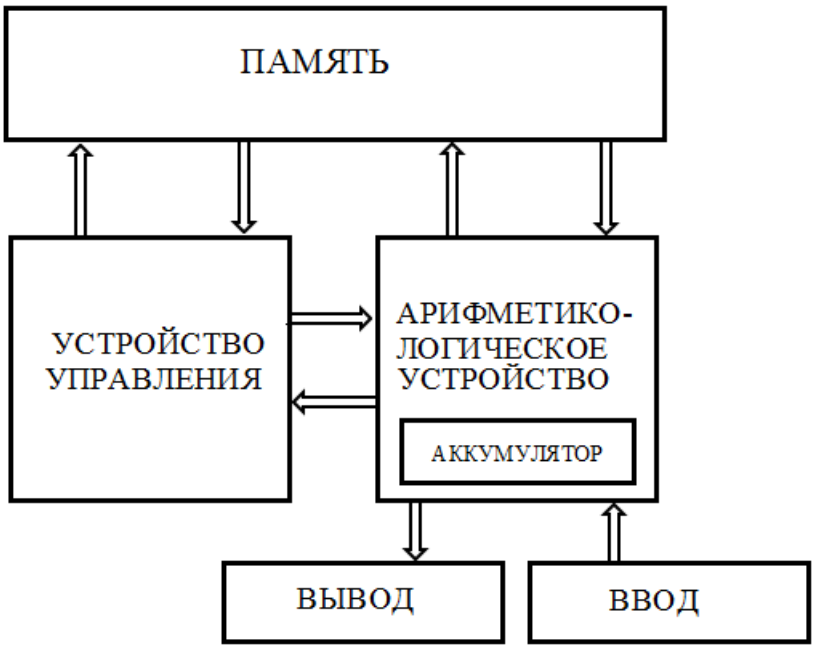


Рис. 2.2. Архитектура фон-Неймана.

Именно на основе этих принципов производились первые два поколения компьютеров. В более поздних поколениях происходили некоторые изменения, хотя принципы фон-Неймана актуальны и сегодня и могут быть сформулированы следующим образом:

- ***Принцип однородности памяти.*** Команды и данные хранятся в одной и той же памяти и внешне в памяти неразличимы. Распознать их можно только по способу использования, то есть одно и то же значение в ячейке памяти может использоваться и как данные, и как команда, и как адрес в зависимости лишь от способа обращения к нему. Это позволяет производить над командами те же операции, что и над числами, и, соответственно, открывает ряд возможностей. Так, циклически изменяя адресную часть команды, можно обеспечить обращение к последовательным элементам массива данных. Такой приём носит название модификации команд и с позиций современного программирования не приветствуется. Более полезным является другое следствие принципа однородности, когда команды одной программы могут быть получены как результат исполнения другой программы. Эта возможность лежит в основе трансляции — перевода текста программы с языка высокого уровня на язык конкретной вычислительной машины.
- ***Принцип адресности.*** Структурно основная память состоит из пронумерованных ячеек, причём процессору в произвольный момент доступна любая ячейка. Двоичные коды команд и данных разделяются на единицы информации, называемые словами, и хранятся в ячейках памяти, а для доступа к ним используются номера соответствующих ячеек — адреса.
- ***Принцип программного управления.*** Все вычисления, предусмотренные алгоритмом решения задачи, должны быть представлены в виде программы, состоящей из последовательности управляющих слов - команд. Каждая команда предписывает некоторую операцию из набора операций, реализуемых вычислительной машиной. Команды программы хранятся в последовательных ячейках памяти вычислительной машины и выполняются в естественной последовательности, то есть в порядке их положения в программе. При необходимости, с помощью специальных команд, эта последовательность может быть изменена. Решение об изменении порядка выполнения команд программы принимается либо на основании анализа результатов предшествующих вычислений, либо безусловно.

Таким образом, сформулированный фон-Нейманов принцип программного управления для современных ЭВМ предполагает следующее уточнение понятия алгоритма:

- исходные данные, промежуточные и конечные результаты кодируются обычно в двоичной форме и разделяются на единицы или элементы информации, называемые словами;
- правила вычислений задаются как композиция операторов двух типов - ***преобразователей*** и ***распознавателей.*** (Преобразователи обеспечивают собственно преобразование информации путем выполнения действий над элементами информации (словами). Распознаватели - обеспечивают управление порядком выполнения операторов путем анализа элементов информации);
- операторы кодируются управляющими словами, которые определяют наименование операции и слова информации (операнды), участвующие в операции. Управляющие слова называются командами.
- разнотипные слова информации (данные и команды) различаются не по способу кодирования, а по способу использования;
- слова информации размещаются в ячейках памяти и идентифицируются номерами ячеек, называемыми адресами слов.

Алгоритм, представленный совокупностью команд называется ***программой***. Выполнение алгоритма сводится к последовательному выполнению команд в порядке, однозначно определенном программой.

Независимость кодов типа информации и их расположение в одной и той же памяти дают возможность модифицировать программу, оперируя управляющими словами как данными (хотя такая модификация не практикуется). Идентификация ячеек последовательными номерами определяет естественным образом единственную структуру данных - линейную, всякие другие структуры, которыми оперирует программист, должны преобразовываться именно к этой структуре. Поэтому процедуры распределения памяти и учета свободного пространства весьма сложны и громоздки.

Конечно, сегодня уже вполне очевидны недостатки модели фон-Неймана, одним из основных которых является совместное использование шины для памяти программ и памяти данных, что и приводит к узкому месту архитектуры фон Неймана, а именно ограничению пропускной способности между процессором и памятью по сравнению с объёмом памяти. Из-за того, что память программ и память данных не могут быть

доступны в одно и то же время, пропускная способность канала «процессор-память» и скорость работы памяти существенно ограничивают скорость работы процессора — гораздо сильнее, чем если бы программы и данные хранились в разных местах.

Сегодня данная проблема решается совершенствованием систем ***кэширования***, что в свою очередь усложняет архитектуру систем и увеличивает риск возникновения побочных ошибок.

Учёные из США и Италии, например, еще в 2015 заявили о создании прототипа ***мем-процессора*** (англ. ***memprocessor***) с отличной от архитектуры фон Неймана и возможности его использования для решения ***NP***-полных задач.

Несмотря на недостатки, ***фон-неймановский принцип*** остается в настоящее время основным принципом программного управления. Однако в последние годы, стараясь достигнуть существенного повышения их производительности, в ряде случаев отходят от модели фон Неймана. Возможно построение машины с отдельной памятью для команд и отдельной памятью для данных, и которая допускает параллельное во времени извлечение их из памяти и передачу по соответствующим шинам. Такая структура (модель) получила название ***гарвардской***, т.к. была реализована впервые в 1944г. в Гарвардском университете в ранней релейной ВМ, предшествующей появлению ЭВМ. Гарвардская модель реализована, в частности, в некоторых микропроцессорах.

Для кодирования информации в настоящее время наиболее широкое распространение получила двоичная система счисления.

Следует подчеркнуть, что для всех используемых множеств различных способов представления данных в ЭВМ характерна неделимость слов информации: слово обрабатывается целиком как самостоятельный информационный объект.

Управляющие слова - ***команды*** - также представляют собой двоичный код, кодирующий элементарный оператор. Команда, соответствующая преобразователям, содержит информацию об операции и операндах. Поэтому слово, представляющее команду, состоит из нескольких частей (полей) - кода операции и кода операндов. ***Код операции*** задает элементарную ***машинную операцию***, как, например, сложение, вычитание, умножение, деление и т.п.). Любая ЭВМ реализует набор операций, который должен обладать свойством полноты (т.е. достаточностью для описания алгоритма) и обеспечивать эффективность обработки данных.

Данные и команды записаны в ячейках памяти. Для обозначения места данных и команд в памяти ЭВМ нет никаких других средств, кроме ***адресов***-чисел, представляющих собой номер ячейки.

В команде должны указываться адреса операндов. Использование в команде адресов вместо самих операндов придает программе универсальность.

Для бинарных операций требуется два операнда, а команде необходимо три адреса: два для указания операндов и один для ячейки результата.

Все эти данные расположены в команде в определенном порядке, задаваемом ***форматом команды*** - схемой расположения двоичных цифр, позволяющей различать ее составные части и определять их функции. ЭВМ имеют несколько форматов команд, отличающихся числом и расположением адресов операндов.

Процесс выполнения программы заключается в последовательности выполнения команд. Т.к. команды записаны в памяти, чтобы начать выполнение программы, надо задать адрес первой команды, а в каждой выполненной команде указать адрес следующей команды или определить способ вычисления адреса следующей команды из адреса выполненной. Обычно предполагается естественный порядок выполнения команд. Последовательно выполняемые команды программы должны быть записаны в последовательные ячейки памяти. Тогда адрес следующей команды будет вычисляться увеличением адреса выполненной команды. Для изменения такого порядка выполнения команд предусматривается специальная команда перехода, в адресной части которой указывается адрес следующей команды.

Распознавателям соответствуют команды условного перехода, в которых указывается условие и адрес перехода. Если полученный в результате выполнения предыдущих команд результат удовлетворяет заданным в команде условного перехода условиям, то выполняется переход по заданному в команде адресу. В противном случае выполняется следующая по порядку команда. Для прекращения выполнения программы должна использоваться специальная команда.

Реализуемая ЭВМ функция целиком определяется программой. Замена программы изменяет и функцию ЭВМ.

Рассмотрим простейшие программы для гипотетической ЭВМ. Все адреса, коды операций и другие числа для удобства будем записывать в десятичной системе счисления

Пример 2.1. Пусть необходимо вычислить значение арифметического выражения $Y=(A+B)/C$. Будем считать, что к моменту пуска программы исходные данные каким-либо образом записаны в память ЭВМ. Кодирование операций задано таблицей 2.1. Тогда программа будет иметь вид, представленный в таблице 2.2.

Таблица 2.1

Код операции	Название Операций
1	Сложение
2	Вычитание
3	Умножение
4	Деление
5	Останов

Программа (табл.2.2) состоит всего из трех команд, расположенных в последовательных адресах 00, 01, 02. По первой команде складываются числа *A* и *B*, записанные соответственно в адресах 03 и 04, и результат сложения записывается в ячейку *Y* с адресом 06. По второй команде выполняется деление содержимого ячейки *Y* (адрес 06) на содержимое ячейки *C* (адрес 05) и результат снова записывается в ячейку *Y*. По третьей команде ЭВМ прекращает вычисления. При этом не требуется операндов.

Таблица 2.2

Адреса ячеек памяти		Содержимое ячеек				Комментарий
		КОП	Адреса операндов			
			1	2	3	
Програ	00	1	03	04	06	$Y=A+B$
мма	01	4	06	05	06	$Y=Y/C$
	02	5	-	-	-	СТОП
Данные	03					*A*
	04					*B*
	05					*C*
	06					*Y*

Пример 2.2. Теперь рассмотрим простейшую программу для гипотетической ЭВМ с использованием ветвлений алгоритма.

Составить программу для вычисления значения *Y*:

$$Y = \begin{cases} (A+B)/C, \text{ если } C \neq 0; \\ A*B, \qquad \text{если } C = 0. \end{cases}$$

Здесь понадобится распознаватель – условный переход. Примем, что он будет иметь код 6, а условия ветвления – как приведено в таблице 2.3.

Таблица 2.3

Код условия	Условие перехода
00	Безусловный Переход (БП)
01	Переход, если < 0 (УП по < 0)
02	Переход, если =0 (УП по = 0)
03	Переход, если > 0 (УП по > 0)

Таблица 2.4

Адреса ячеек памяти		Содержимое ячеек				Комментарий
			Адреса операндов			
		КОП	1	2	3	
Программа	00	2	09	11	10	*Y*=*C*-0
	01	6	02	05	-	УП по = 0
	02	1	07	08	10	*Y*=*A*+*B*
	03	4	10	09	10	*Y*=*Y*/*C*
	04	5	-	-	-	
	05	3	07	08	10	*Y*=*A*·*B*
	06	6	00	04	-	БП
Данные						
	07					*A*
	08					*B*
	09					*C*
	10					*Y*
	11	0	00	00	00	0 (нуль)

Программа вычислений приведена в таблице 2.4. По команде, записанной по адресу 00, из *C* (адрес 09) вычитается нуль (адрес 11). Результат записывается в ячейку *Y* (адрес 10). При выполнении этой операции вырабатывается и запоминается признак результата, который равен 01, 02 или 03, если полученный результат меньше нуля, равен нулю или больше нуля соответственно. По команде ветвления или условного перехода (УП) переход совершается, если значение признака результата совпадает со значением, указанным в команде перехода. В данном случае, если в результате вычитания *C*-0 выработался признак 2 (C=0), то следующая команда будет взята по адресу 05, в противном случае – из

адреса 02. В первом случае *Y* присвоится значение *A-B*, а во втором – *(A+B)/C*.

В случае, когда нужно изменить последовательность выборки команд из памяти, применяется безусловный переход. В программе эта команда записана по адресу 06.

Заметим, что вместо безусловного перехода можно записать вторую команду останова.

2.2 Структурная организация ЭВМ.

Из неймановского принципа программного управления однозначно определяется состав ЭВМ. Основные устройства, без которых вообще невозможно реализация принципа программного управления, это память и процессор. Назначение памяти - хранение программ и данных, процессора - реализация программного управления, т.е. выборка команд из памяти и их выполнение. Процессор должен иметь внутри себя память небольшой емкости для хранения выполняемой команды, операндов, а также адресов команд и операндов и арифметико-логическое устройство (АЛУ), выполняющее сами операции. Совместную работу всех узлов, блоков и устройств, входящих в процессор, организует устройство управления.

Функционирование процессора происходит в следующем порядке:

1. выборка из памяти команды и вычисление адреса следующей команды;
2. анализ кода операции и определение ее типа, количества операндов и т.п. (первичная дешифрация команды);
3. вычисление адресов и выборка из памяти операндов;
4. вторичная дешифрация и выполнение команды.

Адрес первой команды (стартовый адрес) передается в процессор извне с помощью органов управления или формируется автоматически. После выборки команды процессор уже сам вычисляет адрес следующей команды.

В ЭВМ применяется большое число разнообразных форматов команд, отличающихся друг от друга кодами операций, расположением и количеством адресов операндов. Основной задачей первичной дешифрации и является распознавание команды, ее формата, длины, и т.д., в том числе определяется тип команды – распознавательная или преобразовательная.

Таким образом, в состав ЭВМ входят устройства трех основных классов:

- ***операционные*** - предназначенные для выполнения операций над информацией;

- ***запоминающие*** - предназначенные для хранения элементов информации;
- ***ввода-вывода*** - предназначенные для связи ЭВМ с окружающей средой.

Основным операционным устройством ЭВМ является процессор.

Запоминающие устройства (ЗУ) разделяются на оперативные (ОЗУ) и внешние (ВЗУ). В ЭВМ применяется многоуровневая память. Непосредственно доступна процессору только информация, хранящаяся в ОЗУ. При необходимости использования информации из ВЗУ делается ее пересылка в ОЗУ. ВЗУ имеет много общего с устройствами ввода-вывода (УВВ), и поэтому также как и УВВ относится к одному классу устройств - ***периферийным*** устройствам (ПУ).

При рассмотрении ЭВМ принято различать их ***архитектуру*** и ***структуру***.

Архитектура ЭВМ - понятие, охватывающее общую логическую организацию ЭВМ, состав и назначение ее функциональных средств, принципы кодирования и т.п., т.е. все то, что однозначно определяет принцип обработки информации на данной ЭВМ.

Структура ЭВМ - совокупность элементов и связей между ними. Совместное функционирование взаимосвязанных между собой элементов, представляемое совокупностью физических процессов, приводит к реализации заданных функций ЭВМ, т.е. к вычислениям на основе алгоритмов.

Ввиду большой сложности современных ЭВМ принято представлять их структуру иерархически, т.е. понятие “элемент” жестко не фиксируется.

2.2.1. Модульный принцип построения ЭВМ

Суть принципа: ЭВМ строится из набора устройств и блоков - модулей, реализующих законченные функции и обладающих свойством независимости от других модулей. В конструктивном отношении модуль также представляет собой законченный конструктивный элемент.

Основное преимущество модульного принципа:

- возможность совершенствовать ЭВМ без изменения ее функциональной организации и даже в процессе эксплуатации путем замены отдельных блоков на новые (более совершенные) или путем добавления новых модулей;
- возможность компоновки из модулей большого количества разных по характеристикам ЭВМ;

- уменьшение времени восстановления работоспособности ЭВМ при отказах за счет упрощения поиска неисправности и ремонта.

Модули соединяются ***шинами***.

Совокупность шин, связывающих два модуля, и алгоритм, определяющий порядок обмена информацией между ними, называется ***интерфейсом*** (***сопряжением- interface, interface module*** - интерфейсный модуль).

Следует заметить, что от типа интерфейса во многом зависит производительность и надежность ЭВМ.

Физически ***шина*** представляет собой совокупность проводников электрического тока. Шины интерфейса в зависимости от назначения можно разделить на:

- информационные (для передачи команд, данных, адресов);
- идентификации типа информации;
- управляющие.

Можно говорить о нескольких аспектах интерфейса: ***функциональном, электрическом, механическом***.

Функциональная совместимость модулей требует выработки определенных управляющих сигналов, генерируемых обменивающимися модулями и необходимых для правильных передач информации.

Электрическая совместимость обеспечивается определенными уровнями сигналов, их мощностями и т.п.

Механическая совместимость предполагает применение определенных типов и размеров конструкций, разъемов.

Интерфейс характеризуют его ширина и скорость обмена информацией. Ширина интерфейса – максимальный размер информационного элемента, который может быть через него передан (измеряется в битах, байтах). Скорость интерфейса – количество единиц информации, передаваемых за единицу времени.

Чтобы обеспечить возможность использования модульного принципа, необходимо стандартизировать интерфейсы модулей определенного класса.

2.2.2 Варианты структур ЭВМ

Наиболее простой и естественный способ соединения устройств (модулей) между собой для построения ЭВМ – ***принцип единого интерфейса,*** **(*магистрально-модульный принцип*)** в соответствии с которым к интерфейсу подключаются все устройства, входящие в состав ЭВМ, как это показано на рис.2.3.

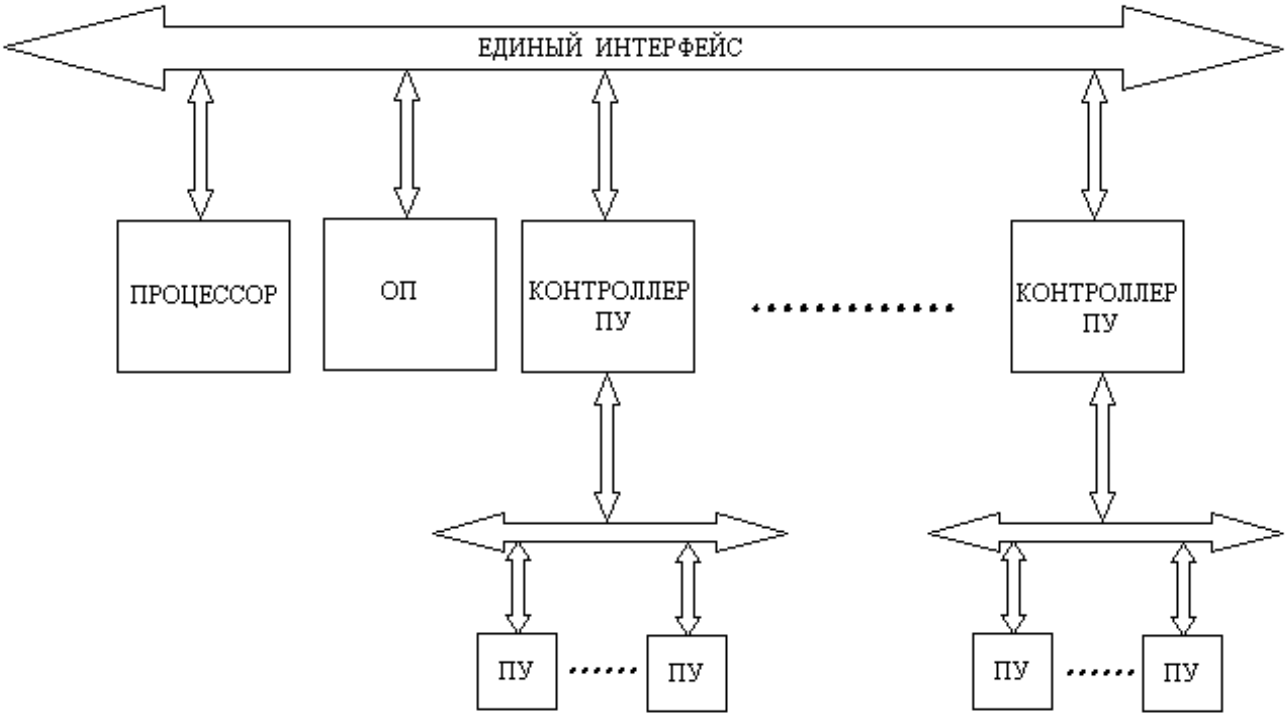


Рис. 2.3
Магистрально-модульный принцип построения ЭВМ

Периферийные устройства (ПУ) подключается к единому интерфейсу через ***контроллер - устройство управления ПУ***. Контроллер ПУ относится к классу операционных устройств и выполняет следующие функции:

- преобразует стандартные последовательности сигналов интерфейса в последовательности сигналов, обеспечивающие работу конкретного ПУ;
- обеспечивает синхронизацию работы ПУ с другими устройствами и в первую очередь с процессором;
- обеспечивает буферизацию информации, т.е. временное хранение в своей внутренней памяти обмениваемых данных на время, необходимое для реализации элементарного акта обмена.

Каждый тип ПУ требует применения специфического контроллера. ПУ подключается к контроллеру с помощью специфичного интерфейса - интерфейса устройства.

Применение единого интерфейса порождает следующие правила обмена информацией:

1. ***Информация передается словами. Ширина интерфейса обычно равна длине слова ОП.***

2. ***В каждый момент времени обмениваются информацией одна пара устройств - источник и приемник информации.***
3. ***Прямой обмен информацией между двумя ПУ невозможен: источником или приемником всегда является процессор или ОЗУ (ОП).***

Если необходимость обмена возникает одновременно в нескольких устройствах, то конфликт между ними разрешается на основе приоритетов с помощью специального устройства - контроллера шин (включается в состав процессора).

Причина обмена информацией между ПУ только через процессор или ОП - различие скоростей работы ПУ, которое приводит к необходимости буферизации.

Принцип единого интерфейса (ЕИ) - высокоэффективный способ организации обмена информацией в ЭВМ, комплектуемых небольшим числом ПУ. Кроме того, в ЕИ обеспечивается большое многообразие режимов обмена и высокая оперативность реакции системы на ситуации во внешней среде.

В качестве примера на рис. 2.4 приведена структура современной магистрально-модульной ЭВМ, построенной по принципу Джон фон Неймана.

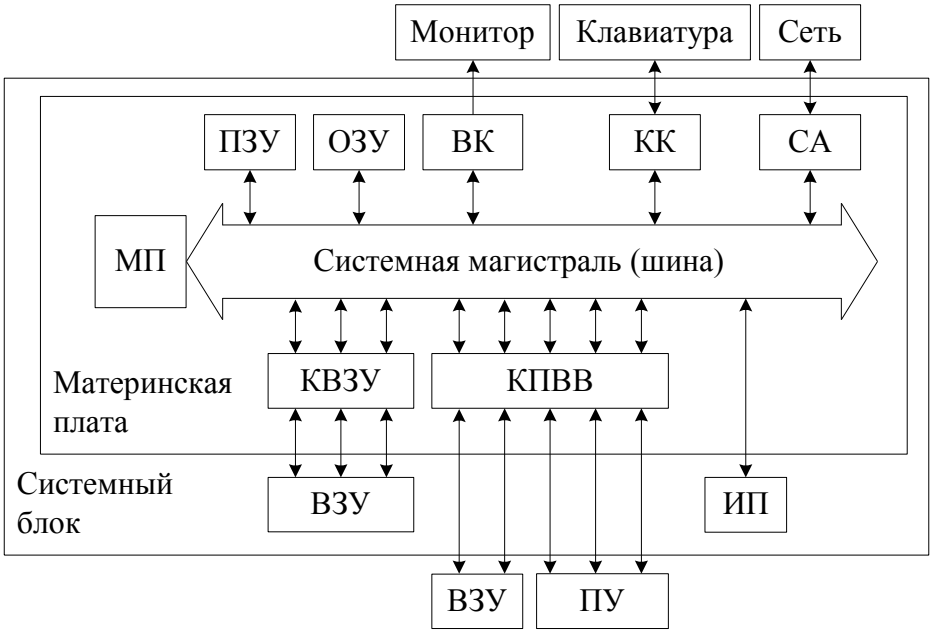


Рис. 2.4 Структура ЭВМ

ЭВМ состоит из ***системного блока***, к которому подключаются монитор и клавиатура. В системном блоке находятся основные компоненты ЭВМ:

- ***ВЗУ – внешние запоминающие устройства*** (жесткий диск, приводы CD/DVD/Blu-Ray, флэш-память); некоторые ВЗУ располагаются внутри системного блока и подключаются к ***контроллерам*** ВЗУ, а некоторые – снаружи системного блока и подключаются к портам ввода-вывода.
  - ***ВК – видеокарта*** (***видеоадаптер, видеоконтроллер***) формирует изображение и передает его на монитор;
  - ***ИП – источник питания*** обеспечивает питание всех блоков ЭВМ по системной шине;
  - ***КВЗУ – контроллеры внешних запоминающих устройств*** управляют обменом информацией с ВЗУ;
  - ***КК – контроллер клавиатуры*** содержит буфер, в который помещаются вводимые символы, и обеспечивает передачу этих символов другим компонентам;
  - ***КПВВ – контроллеры портов ввода-вывода*** управляют обменом информацией с периферийными устройствами;
  - ***МП – микропроцессор*** выполняет команды программы, управляет взаимодействием всех компонент ЭВМ;
  - ***ОЗУ – оперативное запоминающее устройство*** хранит исходные данные и результаты обработки информации во время функционирования ЭВМ;
  - ***ПЗУ – постоянное запоминающее устройство*** хранит программы, выполняемые во время загрузки ЭВМ;
  - ***ПУ – периферийные устройства*** различного назначения: принтеры, сканнеры, манипуляторы «мышь» и др.;
  - ***СА – сетевой адаптер*** (карта) обеспечивает обмен информацией с локальными и глобальными компьютерными сетями.

К устройствам ввода информации относят клавиатуру и такие ПУ, как сканнеры, манипуляторы типа «мышь», джойстики, а к устройствам вывода информации – монитор и такие ПУ, как принтеры.

ЭВМ с канальной организацией.

При комплектации ЭВМ большим числом ПУ, а также при необходимости массовой обработки информации использовалась более сложная иерархическая структура - ***ЭВМ с канальной организацией***, которая применялась в машинах III^{го} и частично машинах IV^{го} поколений**.** В этой структуре к ОЗУ подключены процессор и каналы ***ввода-вывода*** (КВВ). Контроллеры ПУ через интерфейс ввода-вывода подключены к каналам. Ширина интерфейса ввода-вывода обычно равна восьми двоичным разрядам и никак не связана с длиной слова памяти (8 бит -

естественная длина слова для большинства ПУ). Функции контроллера ПУ в основном остаются такими же, как и рассмотренные выше в системе единого интерфейса, но общее для всех контроллеров оборудование вынесено в канал, а в контроллерах оставлены только схемы, специфичные для конкретного типа ПУ.

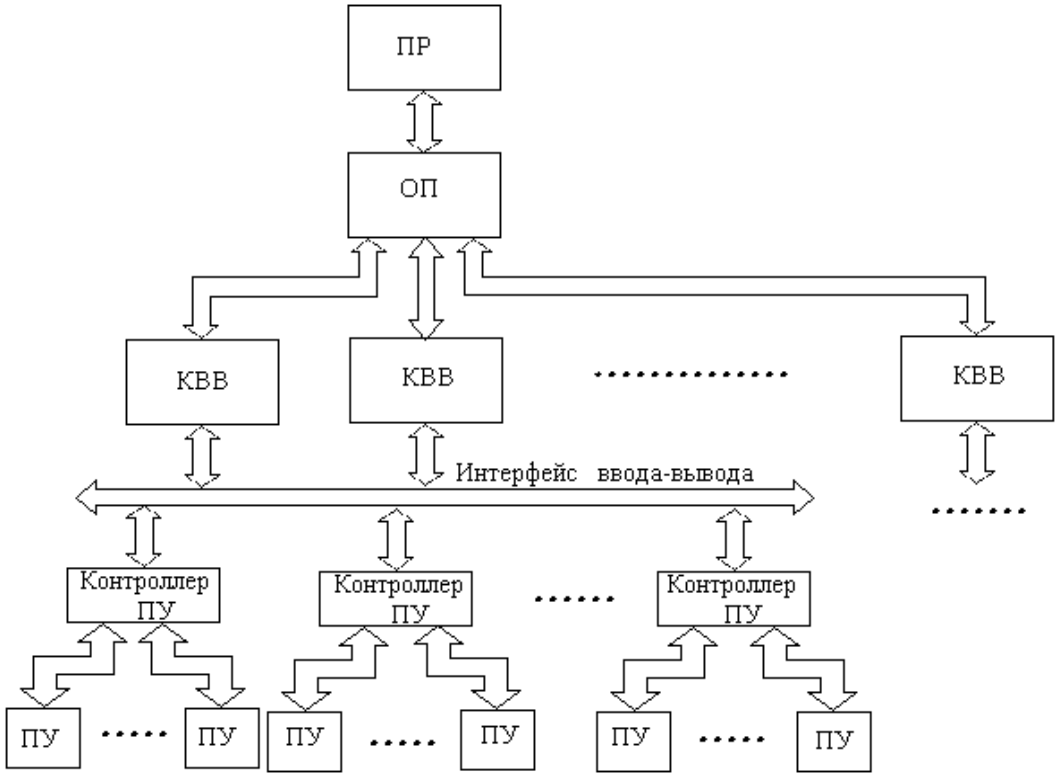


Рис 2.5 Структура ЭВМ с канальной организацией

Основная характерная черта КВВ состоит в том, что он работает по хранимой в памяти программе на основе ППУ, т.е. так же, как и процессор. Но система команд канала приспособлена для выполнения операций обмена. По сути, КВВ - специализированный процессор ввода-вывода. Это позволяет полностью освободить центральный процессор от обслуживания операций обмена ПУ с ОП. Типовая структурная схема организации ЭВМ с канальной организацией приведена на рис.2.5.

В ЭВМ с канальной организацией используются интерфейсы четырех типов:

- Оперативной памяти (через интерфейс осуществляется обмен информацией между ОП, процессором и каналами);
- Процессор-канал, необходимый для обмена управляющими сигналами между ними;
- Ввода-вывода (через интерфейс контроллеры ПУ подключаются к каналу);

- Устройств (с помощью интерфейса ПУ подключаются к контроллеру).

В общем виде обмен информацией выполняется в следующем порядке:

1. Процессор, получив команду ввода-вывода, передает ее в канал;
2. Канал из фиксированной ячейки памяти выбирает начальный адрес канальной программы и начинает ее выполнять, каждая команда начальной программы задает параметры одной операции обмена - направление обмена, начальный адрес ОЗУ, число слов и т.п.
3. Канал, выполняя команды обмена, инициирует работу ПУ, последовательно читает или записывает слова информации, обращаясь в ОП, при выводе разделяет слова информации на 8-битные элементы, а при вводе соединяет 8-битные элементы в слова, подсчитывает число слов, завершает обмен.

При большом числе ПУ использование КВВ экономит оборудование за счет централизации в канале весьма сложных функций по обслуживанию ПУ. В то же время ПУ “отдаляются” от процессора и он уже не может столь оперативно реагировать на различие ситуации во внешней среде. Такая структура не выгодна и в случае, когда обмен осуществляется небольшими порциями информации.

ЭВМ с шинной организацией

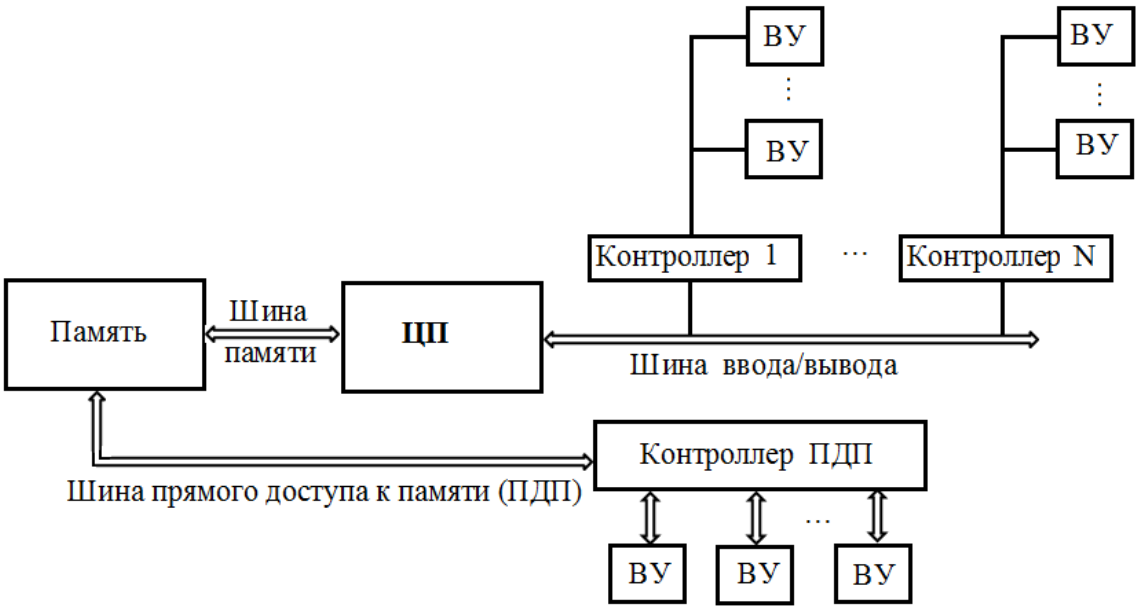


Рис. 2.6. ЭВМ с шинной организацией.

Сегодня заслуживает внимание структура ЭВМ ***с шинной организацией***. Данная организация ЭВМ предложена для разгрузки шины, связывающей процессор с памятью, и как следствие - повышения производительности и надежности работы ЭВМ.

При такой организации используются различные магистрали для связи ЦП с памятью и с внешними устройствами (ВУ). Соответственно, используются различные адресные пространства для обращения к памяти и ВУ. Это требует выделения специальной группы команд ввода-вывода в системе команд процессора. Структура такой ЭВМ приведена на рис. 2.6. Следует отметить, шина прямого доступа к памяти (ПДП) используется для связи ВУ и памяти без участия процессора.

2.3 Технико-эксплуатационные характеристики ЭВМ

Основным техническим параметром ЭВМ является ее ***быстродействие. Быстродействие ЭВМ*** - среднестатистическое число операций (кроме операций ввода, вывода и обращения к внешним запоминающим устройствам), выполняемых вычислительной машиной в единицу времени (номинальное быстродействие). Быстродействие является одним из основных параметров ЭВМ, характеризующий её ***производительность.***

Различают следующие виды быстродействия:

- ***пиковое*** (предельное) - это быстродействие процессора без учета времени обращения к оперативной памяти (ОП) за операндами;
- ***номинальное*** - быстродействие процессора с ОП;
- ***системное*** - быстродействие базовых технических и программных средств, входящих в комплект поставки ЭВМ;

Методы определения быстродействия разделяются на три основных группы:

- расчетные, основанные на информации, получаемой теоретическим или эмпирическим путем;
- экспериментальные, основанные на информации, получаемой с использованием аппаратно-программных измерительных средств;
- имитационные, применяемые для сложных ЭВМ.

Для каждого вида быстродействия применяются следующие традиционные методы их определения.

Пиковая производительность (быстродействие) определяется средним числом команд типа «***регистр-регистр***», выполняемых в одну секунду без учета их статистического веса в выбранном классе задач.

Номинальная производительность (быстродействие) определяется средним числом команд, выполняемых подсистемой «***процессор-память***» с учетом их статистического веса в выбранном классе задач. Она рассчитывается, как правило, по формулам и специальным методикам, предложенным для процессоров определенных архитектур, и измеряется с помощью разработанных для них измерительных программ, реализующих соответствующую эталонную нагрузку.

Для данных типов производительностей используются следующие единицы измерения:

MIPS (***Mega Instruction Per Second***) - миллион целочисленных операций в секунду;

MFLOPS (***Mega Floating Operations Per Second***) - миллион операций над числами с плавающей запятой в секунду;

GFLOPS (***Giga Floating Operations Per Second***) - миллиард операций над числами с плавающей запятой в секунду;

TFLOPS (***Tera Floating Operations Per Second***) - триллион операций над числами с плавающей запятой в секунду.

Системная производительность измеряется с помощью синтезированных типовых (тестовых) оценочных программ, реализованных на унифицированных языках высокого уровня. Унифицированные тестовые программы используют типичные алгоритмические действия, характерные для реальных применений, и штатные компиляторы ЭВМ. Они рассчитаны на использование базовых технических средств и позволяют измерять производительность для расширенных конфигураций технических средств. Результаты оценки системной производительности ЭВМ конкретной архитектуры приводятся относительно базового образца, в качестве которого используются ЭВМ, являющиеся промышленными стандартами систем ЭВМ различной архитектуры. Результаты оформляются в виде сравнительных таблиц, двумерных графиков и трехмерных изображений.

Очень часто при сравнении компьютеров пользуются отношением производительности к стоимости.

Другая не менее важная техническая характеристика ЭВМ - ***ёмкость памяти*** - определяется максимальным количеством данных, которые могут в ней храниться. Ёмкость измеряется в двоичных единицах (***битах***), машинных словах, но большей частью в ***байтах***. Часто ёмкость памяти выражают через число ***К*** = **1024.** Т.о. для измерения ёмкости различных типов запоминающих устройств используют следующие обозначения:

- 1***байт*** = 8 ***бит;***

- 1***Кбит*** (***килобит***) = 1024 ***бит*** = 2^{10} ***бит;***
- 1***Кбайт*** (***килобайт***) = 1024 ***байт*** = 2^{10} ***байт;***
- 1 ***Мбайт*** (***Мегабайт***) = 1024***Кбайт*** = 2^{20} ***байт;***
- 1 ***Гбайт*** (***гигабайт***) = 1024 ***Мбайт*** = 2^{30} ***байт;***
- 1 ***Тбайт*** (***терабайт***) = 1024 ***Гбайт*** = 2^{40} ***байт;***
- 1 ***Пбайт (петабайт)*** = 1024 ***Тбайт*** = 2^{50} ***байт.***

К другим технико-эксплуатационным характеристикам ЭВМ относятся:

- разрядность обрабатываемых слов и кодовых шин интерфейса;
- типы системного и локальных интерфейсов;
- тип и ёмкость оперативной памяти;
- тип и ёмкость накопителя на жестком магнитном диске;
- тип и ёмкость накопителя на гибком магнитном диске;
- тип и ёмкость кэш-памяти;
- тип видеоадаптера и видеомонитора;
- наличие средств работы в компьютерной сети;
- наличие и тип программного обеспечения;
- надежность ЭВМ (свойство ЭВМ при заданных условиях выполнять требуемые функции в течение заданного периода времени);
- стоимость;
- габариты и масса.

2.4. Классификация ЭВМ

Существует достаточно много систем классификации компьютеров. Наиболее часто в технической литературе упоминаются классификация по принципу действия, назначению, вычислительной мощности, уровню специализации, типоразмерам и совместимости. По принципу действия вычислительные машины делятся на три больших класса:

- ***аналоговые*** (АВМ);
- ***цифровые*** (ЦВМ);
- ***гибридные*** (ГВМ).

Аналоговые вычислительные машины -это вычислительные машины непрерывного действия, обрабатывающие информацию, представленную в непрерывной аналоговой форме – в виде непрерывного ряда значений какой-либо физической величины, например, электрического напряжения.

Цифровые вычислительные машины -это вычислительные машины дискретного действия, т.е. работающие с информацией, представленной в дискретной, цифровой форме.

Гибридные вычислительные машины -это вычислительные машины комбинированного действия, работающие с информацией, представленной и в цифровой, и в аналоговой форме. Они совмещают в себе достоинства АВМ и ЦВМ. Гибридные ЭВМ целесообразно использовать для решения задач управления сложными техническими комплексами.

По назначению ЭВМ также разделяются на три группы:

- ***универсальные*** (общего назначения);
- ***проблемно-ориентированные***;
- ***специализированные***.

Универсальные ЭВМ предназначены для решения различных инженерно-технических задач, отличающихся сложностью алгоритмов и большим объемом обрабатываемых данных.

Проблемно-ориентированные ЭВМ служат для решения более узкого круга задач, связанных, как правило, с управлением технологическими объектами, регистрацией, накоплением и обработкой относительно небольших объемов данных.

Специализированные ЭВМ используются для решения узкого круга задач или реализации строго определенной группы функций.

По вычислительной мощности ЭВМ подразделяются на:

- ***сверхбольшие*** (***Супер-ЭВМ***);
- ***большие ЭВМ***;
- ***малые*** (***мини-ЭВМ***);
- ***сверхмалые*** (***микро-ЭВМ***);
- ***персональные компьютеры*** (***ПК***).

Супер-ЭВМ - мощные, высокоскоростные вычислительные машины (системы) с производительностью от сотен миллионов до триллионов операций с плавающей точкой в секунду. Супер-ЭВМ выгодно отличаются от больших универсальных ЭВМ по быстродействию числовой обработки, а от специализированных машин, обладающих высоким быстродействием в сугубо ограниченных областях, возможностью решения широкого класса задач с числовыми расчетами.

При производительности порядка нескольких GFLOPS можно еще обойтись одним векторно-конвейерным процессором (***однопроцессорные Супер-ЭВМ***). Создание высокопроизводительной Супер-ЭВМ с быстродействием порядка TFLOPS по современной технологии на одном процессоре не представляется возможным. Это связано с ограничением, обусловленным конечным значением скорости распространения электромагнитных волн ($3 \cdot 10^8$м/сек), так как время распространения сигнала на расстояние нескольких миллиметров (линейный размер

стороны микропроцессора) при быстродействии, например, 10^{14} оп/с становится соизмеримым с временем выполнения одной операции. Поэтому Супер-ЭВМ с такой производительностью создаются в виде высокопараллельных многопроцессорных вычислительных систем.

В настоящее время в мире насчитывается несколько тысяч Супер-ЭВМ. Рейтинг Top500 производительности суперкомпьютеров составляется дважды в год по результатам выполнения теста Linpack – решению методом Гаусса системы линейных уравнений с плотно заполненной матрицей коэффициентов. Например, мощнейший из суперкомпьютеров семейства IBM Blue Gene/Q Sequoia (США) имеет производительность 17,17 петафлопс, оперативную память - 1,57 петабайт, насчитывает 96 стоек, 1,57 миллионов процессорных ядер. Sequoia был разработан специально для Национальной администрации ядерной безопасности (NNSA), которой требовался высокопроизводительный компьютер для вполне конкретной цели -моделирования ядерных взрывов.

На конец 2020 года самым мощным компьютером являлся суперкомпьютер ***Summit*** фирмы IBM с производительностью 148.6 петафлопс и имеющей в своей структуре 2 414 592 ядра. Расположен в национальной лаборатории Ок-Риджа, США (рис.2.7).



Рис.2.7. Супер-ЭВМ Summit. Процессорный блок (фото).

Summit может обеспечить 200 петафлопс в пике. Это эквивалентно 200 квадриллионам операций с плавающей запятой в секунду.

Это также третий по величине энергосберегающий суперкомпьютер в мире с зарегистрированной эффективностью энергопотребления 14,66 гигафлопс на ватт. На 4600+ серверах Summit, занимающих два баскетбольных поля, размещено более 9 200 процессоров IBM Power 9 и более 27 600 графических процессоров NVIDIA Tesla V100. Система соединена волоконно-оптическим кабелем длиной 298 км и потребляет энергии, достаточной для обеспечения более 10 000 домов. В 2018 году Summit стал первым суперкомпьютером, преодолевшим эксафлопный барьер. Анализируя геномные данные, он достиг пиковой пропускной способности в 1,88 эксафлопс, что составляет почти 2 миллиарда миллиардов вычислений в секунду.

Большие ЭВМ часто называют ***мэйнфреймами*** (***Mainframe***). Они поддерживают многопользовательский режим работы (обслуживают одновременно от десятков до несколько тысяч пользователей).

Основные направления эффективного применения мэйнфреймов - это решение научно-технических задач, работа в вычислительных системах с пакетной обработкой информации, работа с большими базами данных, управление вычислительными сетями и их ресурсами. Последнее направление - использование мэйнфреймов в качестве больших серверов вычислительных сетей - часто отмечается специалистами среди наиболее актуальных.

Примерами больших ЭВМ может служить семейство больших машин ЕС ЭВМ, IBM ES/9000 (1990г.), IBM S/390 (1997г.), а также японские компьютеры M1800 фирмы Fujitsu.

Мини - ЭВМ (***малые ЭВМ***) - надежные, недорогие и удобные в эксплуатации компьютеры, обладающие несколько более низкими по сравнению с мэйнфреймами возможностями. В многопользовательском режиме поддерживаются 16 - 512 пользователей.

Основные их особенности:

- широкий диапазон производительности в конкретных условиях применения,
- аппаратная реализация большинства системных функций ввода-вывода информации,
- простая реализация многопроцессорных и многомашинных систем,
- высокая скорость обработки прерываний,
- возможность работы с форматами данных различной длины.

К достоинствам мини-ЭВМ можно отнести:

1) специфическую архитектуру с большой модульностью;
2) лучшее, чем у мэйнфреймов, соотношение производительность/ стоимость;
3) широкая номенклатура периферийных устройств;
4) повышенную точность вычислений.

Мини-ЭВМ успешно применяются:

- в качестве управляющих вычислительных комплексов в многопользовательских вычислительных системах;
- в системах автоматизированного проектирования;
- в системах моделирования и искусственного интеллекта.

Одними из первых мини-ЭВМ были компьютеры PDP-11 фирмы DEC (США), Система Малых ЭВМ (СМ ЭВМ): СМ1, 2,3,4,1400, 1700 и др. В настоящее время семейство мини-ЭВМ включает большое число моделей от VAX-11 до VAX 8000, супермини-ЭВМ класса VAX 9000 и др.

Появление класса ***микро-ЭВМ*** в 70-х годах изначально определялось использованием в его структуре микропроцессора (МП), изобретенного в 1969 году. Сейчас микропроцессоры используются во всех классах ЭВМ без исключения.

Микро-ЭВМ по назначению можно разделить на ***универсальные*** и ***специализированные*** (рис. 2.8).

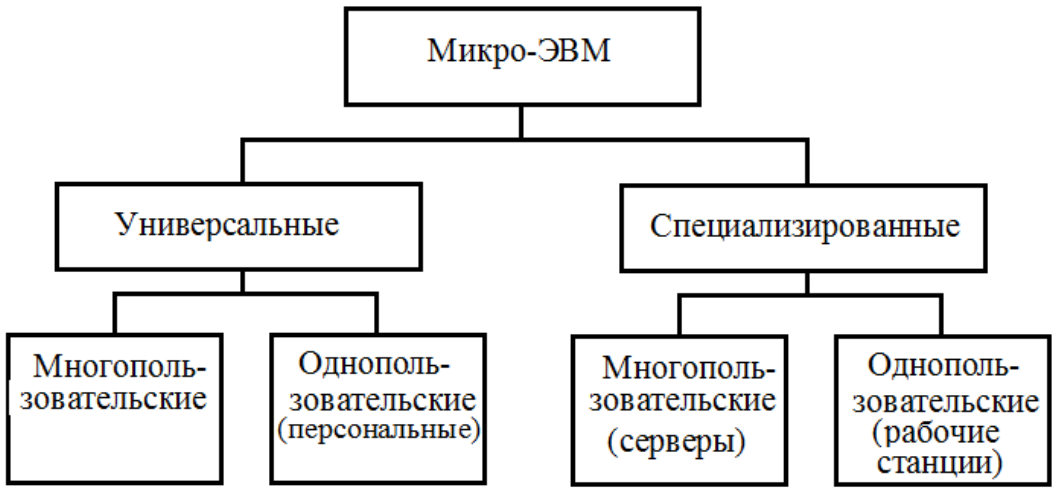


Рис. 2.8. Классификация микро-ЭВМ

Универсальные многопользовательские ЭВМ - это мощные микро-ЭВМ, используемые в компьютерных сетях, оборудованные несколькими видеотерминалами и функционирующие в режиме разделения времени, что позволяет эффективно работать на них сразу нескольким пользователям. Это универсальные серверы (***Server***) компьютерных сетей, обрабатывающие запросы от всех станций сети, выделенный для

обработки запросов от всех станций вычислительной сети, предоставляющий этим станциям доступ к общим системным ресурсам (вычислительным мощностям, базам данных, библиотекам программ, принтерам, факсам и др.) и распределяющий эти ресурсы.

Эту интенсивно развивающуюся группу компьютеров обычно относят к микро-ЭВМ, но по своим характеристикам мощные серверы скорее можно отнести к малым ЭВМ и даже к мэйнфреймам, а супер-серверы приближаются к Супер-ЭВМ.

Универсальные однопользовательские ЭВМ или ***персональные компьютеры*** (***ПК)*** (***Personal Computer***, ***PC***) должны удовлетворять требованиям общедоступности и универсальности применения и иметь следующие характеристики:

- малую стоимость, находящуюся в пределах доступности для индивидуального покупателя;
- автономность эксплуатации без специальных требований к условиям окружающей среды;
- гибкость архитектуры, обеспечивающую ее адаптивность к разнообразным применениям в сфере управления, науки, образования, в быту;
- «дружественность» операционной системы и прочего программного обеспечения для пользователя;
- высокую надежность работы (более 5000 ч. наработки на отказ).

Новый стандарт устанавливает следующие категории персональных компьютеров:

- ***Consumer PC*** (массовый ПК);
- ***Office PC*** (деловой ПК);
- ***Mobile PC*** (портативный ПК);
- ***Workstation PC*** (рабочая станция);с
- ***Entertainment PC*** (развлекательный ПК).

Существует условная классификация ПК ***по типоразмерам***:

- настольные компьютеры (***desktop***);
- настольные мини-компьютеры (***slim-desk***);
- портативные (***notebook***);
- карманные (***palmtop***) модели.

В области персональных компьютеров сегодня наиболее широко распространены две ***аппаратные платформы*** - ***IBM PC*** и ***Apple Macintosh***. Принадлежность компьютеров к одной и той же аппаратной платформе повышает совместимость между ними, а принадлежность к разным платформам -понижает, зачастую до полной несовместимости.

Кроме аппаратной совместимости существуют и другие виды совместимости:

- ***совместимость на уровне операционной системы;***
- ***программная совместимость;***
- ***совместимость на уровне данных.***

Специализированные ЭВМ ориентированы на решение определенного (постоянного) класса задач в течение периода своей эксплуатации. Ориентация специализированных ЭВМ осуществляется различными способами:

- специальной аппаратурной организацией самих ЭВМ или их внешних связей;
- созданием для ЭВМ специального программного обеспечения;
- введением дополнительных аппаратных блоков, расширяющих те или иные функции, возлагаемые на ЭВМ;
- и др.

Сферы использования таких ЭВМ как в нашей стране, так и за рубежом имеют устойчивую тенденцию к расширению. Можно выделить следующие основные области применения специализированных ЭВМ:

1. промышленное производство и транспорт;
2. военная техника и оборона;
3. непромышленная сфера.

Примером специализированных однопользовательских микро-ЭВМ, ориентированных для выполнения определенного круга задач (графических, инженерных, издательских и др.), являются рабочие станции (***workstation***) - однопользовательские системы с мощным процессором и многозадачной ОС, имеющие развитую графику с высоким разрешением, большую дисковую и оперативную память и встроенные сетевые средства.

Специализированные многопользовательские микро-ЭВМ (спец. серверы) осуществляющие управление базами и архивами данных, многопользовательскими терминалами, поддерживающими факсимильную связь, электронную почту и др.

Специализированные серверы используются для устранения наиболее «узких» мест в работе сети, а именно:

- создания и управления базами и архивами данных;
- поддержка многоадресной факсимильной связи и электронной почты;
- управления многопользовательскими терминалами (принтером, плоттером;

- и др.

Файл-сервер используется для работы с файлами данных, имеет объемные дисковые ЗУ.

Архивационный сервер (сервер резервного копирования) предназначен для резервного копирования информации.

Факс-сервер, почтовый сервер - выделенные компьютеры для организации эффективной многоадресной факсимильной связи или электронной почты.

Встраиваемые микро-ЭВМ входят составным элементом в промышленные и транспортные системы, технические устройства и аппараты, бытовые приборы. Они способствуют существенному повышению их эффективности функционирования, улучшению технико-экономических и эксплуатационных характеристик.

Лекция 3.

3. Основы теории информации

3.1 Информация

Информация - любые сведения, являющиеся объектом хранения, преобразования и передачи.

В широком смысле слова ***информация*** является отражением реального мира. Информация - это единственный неубывающий ресурс жизнеобеспечения. Более того: ее объем в настоящее время удваивается ежегодно. Информация, подготовленная для обработки на компьютерах, называется ***данными***. Информационный процесс включает в себя такие этапы:

1. сбор информации от различных источников и представление ее в форме, необходимой для ввода в компьютер;
2. передачу (пересылку) информации от источника к приемнику;
3. хранение — процесс передачи информации во времени;
4. обработку — систематическое выполнение операций над данными;
5. выдачу результата обработки пользователю.

На всех этих этапах используют средства компьютерной схемотехники. К информации предъявляют следующие требования:

- корректность (однозначность восприятия);
- ценность (полезность) и оперативность (актуальность);
- точность, достоверность и устойчивость (способность реагировать на изменения исходных данных);
- достаточность (полнота) — наличие минимально необходимого объема информации для принятия правильного решения.

Структуру и общие свойства информационных процессов изучают в информатике, которая включает:

- теорию информации;
- алгоритмические, программные и компьютерные средства обработки информации;
- архитектуру компьютеров, системы искусственного интеллекта, вычислительные сети и т.д.

В теории информации изучают процессы передачи, преобразования и хранения информации, в том числе:

- методы определения количества информации в сообщении;
- рациональные способы представления информации с помощью различных символов (букв, цифр и т.д.);

- способы формирования, обнаружения и оценки параметров информационных процессов.

Упорядоченную последовательность символов (букв, цифр, математических знаков, предназначенных для передачи информации), закодированную в материальной форме, называют сообщением. Информационное сообщение всегда связано с источником и приемником информации, соединенных каналом передачи (рис. 3.1).



Рис. 3.1. Информационная модель канала передачи

Источником и приемником информации могут быть как люди, так и технические устройства (компьютеры, датчики, индикаторы и др.). Каналом передачи (связи) называется совокупность устройств, имеющих один вход и один выход, предназначенных для передачи информации на расстояния. Сообщения могут иметь различные формы: звука, текста, изображения, электрического напряжения от датчиков (например, от термопар). Информационные сообщения размещают на машинных носителях информации. ***Носитель информации*** - это любая запоминающая предметная среда, предназначенная для записи и хранения информации с целью ее непосредственного ввода в компьютер. Носитель информации является промежуточным звеном между компьютером и первичным документом, содержащим числовые данные, текстовые материалы, схемы, графики, различные измерения. Хранение сообщений - это, как правило, три вида операций: запись, собственно хранение и считывание. Информация записывается в носитель посредством изменения физических или механических свойств запоминающей среды. Данные считываются посредством преобразования их в электрические сигналы. Считают, что в носителе информации сигнал хранится в закодированной форме. Носители информации различаются по следующим признакам:

- средой накопления: непрерывные (магнитные ленты и диски) и дискретные, где каждой хранимой единице данных отводится свой дискретный участок или элемент (ферритовые сердечники, перфокарты, перфоленты, триггеры, криотроны и т.д.);
- типом материала: бумага с текстом или рисунком; бумажные перфоленты, перфокарты; магнитные пленки, магнитные ленты и диски;

- способом считывания данных: механические, оптические, магнитные, электрические;
- конструктивным исполнением: ленточные, дисковые, электронные и др.

В теории передачи и преобразования информации установлены информационные меры количества и качества информации - ***семантические, структурные, статистические***.

Семантический подход позволят выделить полезность или ценность информационного сообщения.

3.1.1 Структурная мера информации

Информация всегда представляется в виде сообщения. Элементарная единица сообщений - символ. Символы, собранные в группы - слова. Сообщение, оформленное в виде слов всегда передается в материально-энергетической среде.

Структурный подход используют для оценки возможностей информационных систем вне зависимости от условий их применения. При использовании структурных мер информации учитывают только дискретное строение сообщения, количество содержащихся в нем информационных элементов, связей между ними.

При структурном подходе различают ***геометрическую, комбинаторную и аддитивную*** меры информации Геометрическая мера определяет параметры геометрической модели информационного сообщения (длина, площадь, объем) в дискретных единицах. Эту меру применяют как для оценки информационной емкости всей модели, так и для оценки количества информации в одном сообщении.

В комбинаторной мере количество информации I определяют количеством комбинаций элементов (символов), которые совпадают с числом:

- сочетаний из q элементов по n; например, для множества цифр 1, 2, 3, 4 можно составить шесть сочетаний по две цифры: 12, 13, 14, 23, 24, 34;
- перестановок $I = q!$; например, для множества букв *а, в, с* можно получить шесть перестановок: *авс, асв, вас, вса, сав, сва*;
- размещений с повторениями из q элементов по n, $I = q^n$. Например, для $q = 0, 1$ и $n = 3$ имеем: 000, 001, 010-, 011, 100, 101, 110, 111.

Аддитивная мера (мера Хартли) - в соответствии с которой количество информации измеряется в двоичных единицах - ***битах*** -

наиболее распространена. Вводится понятие ***глубины q числа*** и ***длины n числа***.

Глубина q числа - количество символов, принятых для представления информации. В каждый момент времени реализуется только один какой-либо символ.

Длина n числа - количество позиций, необходимых и достаточных для представления чисел заданной величины.

При заданной глубине и длине числа количество возможных сообщений длины *n* равняется числу размещений с повторениями

$$N = q^n. \quad (3.1)$$

Эту меру наделяют свойством аддитивности, чтобы она была пропорциональна длине сообщения и позволяла складывать количество информации ряда источников сообщений. Хартли предложил логарифмическую функцию как меру количества информации, определяемую как количество информации при минимальной глубине и минимальной длине числа, т.е. при g=2 и n=1, которая получила название - ***бит***.

$$I(g) = \log_2 N = n \log_2 g \quad (3.2)$$

Следовательно, **1 *бит*** информации соответствует одному символу двоичной системы счисления. С другой стороны, как будет видно далее, 1 бит соответствует элементарному событию, которое может произойти или не произойти. Такая мера количества информации удобна тем, что она обеспечивает возможность оперировать мерой как числом. Количество информации при этом эквивалентно количеству двоичных символов.

При наличии нескольких источников информации общее количество информации определяется как их сумма:

$$I(g_1, g_2, \cdots g_n) = \sum_{i=1}^{n} I(g_i) \quad (3.3)$$

3.1.2 Статистическая мера информации

В статистической теории информации вводится более общая мера количества информации, в соответствии с которой рассматривается не само событие, а информация о нем[2].

Как показал К. Шеннон, в теории информации любое событие, с которым мы имеем дело, представляет собой совокупность сведений о некоторой физической системе. Очевидно, если бы состояние физической системы было бы известно заранее, не было бы смысла передавать сообщения.

[2] К. Шеннон “Избранные труды по теории информации”.

Рассмотрим некую гипотетическую систему X, которая может оказаться в том или ином состоянии, т.е. систему с заведомо присущей ей степенью неопределенности.

$$X \to X_1, X_2, \cdots X_i \qquad (3.4)$$

Очевидно, сведения, получаемые о системе, будут тем больше и ценнее, чем больше была неопределенность системы до получения этих сведений (априори). Возникает естественный вопрос, можно ли измерить степень неопределенности системы. Чтобы ответить на этот вопрос, рассмотрим три следующих примера:

1. Возьмем систему из одной монеты. В результате бросания монеты она может оказаться в одном из двух состояний;
2. Система из игральной кости. В результате бросания кости она может оказаться в одном из шести возможных состояний (“1”, “2”, “3”, “4”, “5”, “6”);
3. Рассмотрим некоторое устройство, которое может находиться в двух состояниях: “исправно”, “неисправно” соответственно с априорными вероятностями 0.99 и 0.01.

Сравнивая первый и второй пример, можно сделать вывод, что во втором случае степень неопределенности больше. Т.е. неопределенность должна оцениваться числом возможных исходов (состояний), в которых может оказаться система.

Сравнивая 1 и 3, можно также сделать вывод, что неопределенность первой системы гораздо больше. Т.е. при одинаковом числе состояний системы неопределенность должна зависеть от вероятности поступления событий.

В качестве меры априорной неопределенности системы в теории информации принимается величина (для дискретных систем)

$$H(x) = -\sum_{i=1}^{n} p(X_i) \cdot \log_a p(X_i) \qquad (3.5)$$

которую Шеннон назвал ***энтропией***, используя формальное ее сходство с энтропией в термодинамике. Основание логарифма влияет лишь на удобство вычисления.

В случае оценки энтропии

1. в двоичных единицах:
   $H(x) = -\sum_{i=1}^{n} p(X_i) \cdot \log_2 p(X_i)$ ***бит/символ***;
2. в десятичных единицах:
   $H(x) = -\sum_{i=1}^{n} p(X_i) \cdot \log p(X_i)$ ***дит /символ*** (1 бит ≈ 0.3 дит);
3. в натуральных единицах:

$H(x) = -\sum_{i=1}^{n} p(X_i) \cdot \ln p(X_i)$ ***нит/символ*** (1 бит ≈ 0.693 нит).

Как видно, 1 бит информации соответствует одному символу двоичной системы счисления, которая повсеместно используется в цифровых устройствах, и в частности –в ЭВМ

Например, для рассмотренных выше примеров получаем:

1. $H(x) = -\sum_{i=1}^{2} p(X_i) \cdot \log_2 p(X_i) = -0.5 \cdot \log_2 0.5 - 0.5 \cdot \log_2 0.5 =$ $= 0.5+0.5 = 1$бит.
2. $H(x) = -\sum_{i=1}^{6} p(X_i) \cdot \log_2 p(X_i) = -6 \cdot \frac{1}{6} \cdot \log_2 \frac{1}{6} = 2.58$ бит.
3. $H(x) = -\sum_{i=1}^{2} p(X_i) \cdot \log_2 p(X_i) = -0.99 \cdot \log_2 0.99 -$ $0.01 \cdot \log_2 0.01 = 0.08079$ бит.

Если все элементы сообщения несут одинаковое количество информации, что часто имеет место в технических системах, то в этом случае энтропия - это количество информации, которое приходится на один элемент сообщения:

$$H = \frac{I}{n} = \log q. \qquad (3.6)$$

Как было уже показано, основание логарифма зависит от выбора единицы количества информации. Если для алфавита используют двоичные цифры 0 и 1, то за основание логарифма принимают q = 2, в результате чего $I = n \log_2 2 = n$. При длине n = 1 получают количество информации I = 1, которую называют ***битом***. Как видно из примера 1, передача сообщения длиной n=1 эквивалентна выбору одного из двух возможных равновероятных сообщений - одно из них равно единице, другое - нулю. Двоичное сообщение длины n содержит n битов информации. Если основание логарифма равно 10, то количество информации измеряется в десятичных единицах - ***дитах***, причем 1 ***дит*** = 3,32 бита. Например, текст составлен из 32 букв алфавита и передается последовательно по каналу связи в двоичном коде. При этом количество информации приходящее на один символ $I = \log_2 32 = \log_2 2^5 = 5$ бит. В общем случае сообщения появляются с разной вероятностью.

Статистическая мера использует вероятностный подход к оценке количества информации. Как было показано, каждое сообщение характеризуется вероятностью появления, и чем она меньше, тем больше в сообщении информации. Вероятность конкретных типов сообщений устанавливают на основе статистического анализа. Пусть сообщения образуются последовательной передачей букв некоторого алфавита $x_1, x_2, \cdots x_i, \cdots x_k$ с вероятностью появления каждой буквы $p(x_1) = p_1$, $p(x_2) = p_2$, $\cdots p(x_i) = p_i$, $\cdots p(x_k) = p_k$, при этом выполняется условие: $p_1 + p_2 + \cdots + p_i + \cdots + p_k = 1$. (Множество с известным распределением элементов называют ***ансамблем***). Согласно Шеннону количество

информации, которое содержится в сообщении x_i, рассчитывают по формуле (3.7)

$$I(x_i) = \log\frac{1}{p_i}. \qquad (3.7)$$

Для абсолютно достоверных сообщений $p_i = 1$, и, следовательно, количество информации $I(x_i) = 0$. При уменьшении значения p_i количество информации увеличивается. Пусть в ансамбле все буквы алфавита $x_1, x_2, \cdots x_i, \cdots x_m$ равновероятны, то есть $p_1, p_2, \cdots p_i, \cdots p_m = \frac{1}{m}$, и статистически независимы. Тогда количество информации в сообщении длиной *n* букв с учетом выражения (3.7)

$$I = \sum_{i=1}^{n} I(x_i) = \sum_{i=1}^{n} \log\frac{1}{p_i} = \log\frac{1}{p_1} + \log\frac{1}{p_2} + \cdots + \log\frac{1}{p_m} = n\log m,$$

что совпадает с мерой Хартли в соответствии с выражениями (3.1) и (3.2).

Согласно Шеннону информация - это снятие неопределенности, что понимают следующим образом. До опыта событие (например, появление буквы x_i) характеризуют малой начальной вероятностью $p_{\text{н}}$, которой соответствует большая неопределенность. После опыта неопределенность уменьшается, поскольку конечная вероятность $p_{\text{к}} > p_{\text{н}}$. Уменьшение неопределенности рассчитывают как разность между начальным $I_{\text{н}}$ и конечным $I_{\text{к}}$ значениями количества информации. Например, для $p_{\text{н}}$= 0.1 и $p_{\text{к}}$ = 1 получим:

$$\Delta I = I_{\text{н}} - I_{\text{к}} = \log_2\frac{1}{p_{\text{н}}} - \log_2\frac{1}{p_{\text{к}}} = \log_2 10 - \log_2 1 = 3.32 \text{ бит}.$$

Пусть сложное сообщение характеризуется алфавитом из букв $x_1, x_2, \cdots x_i, \cdots x_k$, с их вероятностями $p_1, p_2, \cdots p_i, \cdots p_k$ и частотой появления каждой буквы $m_1, m_2, \cdots m_i, \cdots m_k$. Все сообщения статистически независимы, при этом $m_1 + m_2 + \cdots + m_i + \cdots + m_k = m$. Общее количество информации для всех k типов сообщений с учетом выражения (3.7)

$$I_{\Sigma} = \sum_{i=1}^{k} m_i \log\frac{1}{p_i}. \qquad (3.8)$$

Среднее значение количества информации на одно сообщение (энтропия) согласно формуле Шеннона (3.7)

$$H = I_{\Sigma} = \sum_{i=1}^{k} \frac{m_i}{m}\log\frac{1}{p_i} = \sum_{i=1}^{k} p_i \log\frac{1}{p_i}, \qquad (3.9)$$

где при большом значении m отношение m_i/m характеризует вероятность p_i каждой буквы. Выражение $\log 1/p_i$ рассматривают как частную энтропию, которая характеризует информативность буквы x_i, а энтропию H — как среднее значение частных энтропий. При малых

значениях p_i частная энтропия велика, а с приближением p_i к единице она приближается к нулю.

3.1.3 Основные свойства энтропии

1. Энтропия есть величина вещественная, ограниченная и неотрицательная
2. Энтропия равна нулю тогда и только тогда, когда все вероятности кроме одной равны нулю и эта единственная вероятность равна единице.
3. Энтропия максимальна, если все состояния равновероятны:

$$H(X) = -\sum_{i=1}^{n} P_i \cdot \log_2 P_i = -n \cdot \frac{1}{n} \cdot \log_2 \frac{1}{n} = \log_2 n. \qquad (3.10)$$

Доказательство:

$H(X) = -\sum_{i=1}^{n} P_i \cdot \log_2 P_i; \quad \varphi(P) = 1 - \sum_{i=1}^{n} P_i; \quad \sum_{i=1}^{n} P_i = 1.$

Найдем условный экстремум методом неопределенных множителей Лагранжа (вариационная задача), для чего составим функционал:

$$\Phi(P, \lambda) = H + \lambda\varphi = -\sum_{i=1}^{n} P_i \cdot \log_2 P_i + \lambda(1 - \sum_{i=1}^{n} P_i);$$

$$\frac{d\Phi(P,\lambda)}{dP_k} = -\log_2 P_k - \log_2 e - \lambda = 0;$$

$$\log_2 P_k = -\lambda - \log_2 e,$$

т.е. P_k не зависит от индекса k, и следовательно $P_1 = P_2 = \ldots = P_n = \frac{1}{n}$, ч.т.д.

В этом случае оценки количества информации по Хартли и Шеннону совпадают.

4. Энтропия объединения статистически независимых сообщений определяется суммой энтропий каждого сообщения (теорема сложения энтропий):

$$H(x,y) = -\sum_i \sum_j P(x_i, y_j) \cdot \log P(x_i, y_j) = \langle P(x_i, y_j) = P(x_i) \cdot P(y_j) \rangle =$$

$$= -\sum_i \sum_j P(x_i) \cdot P(y_j) \cdot [\log P(x_i) + \log P(y_j)] =$$

$$= -\sum_i P(x_i) \cdot \log_2 P(x_i) \cdot \sum_j P(y_j) - \sum_i P(x_i) \cdot \sum_j P(y_j) \cdot \log_2 P(y_j) =$$

$$= -\sum_i P(x_i) \cdot \log_2 P(x_i) - \sum_j P(y_j) \cdot \log_2 P(y_j) = H(x) + H(y). \qquad (3.11)$$

5. Энтропия объединения статистически зависимых сообщений определяется через условную энтропию одной из систем.

В этом случае $P(x_i, y_j) = P(x_i) \cdot P(\frac{y_j}{x_i})$, и

$$H(x,y) = H(x) + H\left(\frac{y}{x}\right) = H(y) + H\left(\frac{x}{y}\right), \qquad (3.12)$$

где $H\left(\frac{x}{y}\right) = -\sum_i \sum_j P(x_i, y_j) \cdot \log_2 P\left(\frac{x_i}{y_j}\right) -$ условная энтропия.

3.2 Количество информации

Количество информации есть неопределенность, снимаемая при получении сообщения, исходя из чего количество информации может быть определено как произведение общего числа сообщений K на среднюю энтропию H, приходящая на одно сообщение:

$$I = K \cdot H(\text{бит}), \tag{3.13}$$

или

$$I = K \cdot \sum_{i=1}^{n} P(x_i) \cdot \log_2 P(x_i) \text{ (бит)}. \tag{3.14}$$

Количество информации может быть также определено как снятая неопределенность при передаче сообщения, т.е.

$$I = H_{apriori} - H_{aposteriori}. \tag{3.15}$$

Это определение является наиболее общим и окончательно разрешает вопрос об общности и различии понятий количества информации и энтропии: энтропия - это понятие первичное (исходное), а количество информации - это понятие вторичное (производное).

Имеется еще одна количественная оценка информации - ***объем информации***, который может быть вычислен как

$$Q = k \cdot l, \tag{3.16}$$

где k - количество переданных символов, l - средняя длина кодовых комбинаций вторичного алфавита. Т.е. ***информационным объемом сообщения*** (объемом информации) называется количество битов в этом сообщении. Подсчет информационного объема сообщения является чисто техническим заданием, так как при таком подсчете содержание сообщения не играет никакой роли.

Пример 3.1. Определить объем информации, содержащийся на 1 странице машинописного текста, если на одной странице умещается по 40 строк и в каждой из которых содержится по 70 символов. Для кодирования символов используется 8-ми разрядный код ASCII.

Решение:

Определим количество символов на одной стороне листа:

40 строк · 70 символов = 2800 символов.

Количество информации = 2800 символов · 1 байт = 2800 байт =

=2800:1024 = 2.74 Кбайт. = 2800·8 =22400 бит = 21.88 Кбит.

Пример 3.2. Какое количество вопросов необходимо задать, чтобы наверняка «угадать» студента из группы численностью 25 человек?

Решение:

$2^n = N$, где $N = 25$. Тогда $n = \log_2 N = 4.64$. Берем ближайшее большее $\log_2 N$, т.е. n=5.

Значит, чтобы определить из 25 человек необходимо получить 5 бит информации, т. е. задать 5 вопросов.

3.3 Основные информационные характеристики

1. ***Скорость передачи информации*** (в отсутствии помех)

$$C = n \cdot H \text{ (бит/сек.)}, \qquad (3.17)$$

где n - количество символов, вырабатываемых источником сообщений в единицу времени; H - энтропия, снимая при получении одного символа сообщений, вырабатываемых источником.

Скорость передачи может быть представлена также следующим образом:

$$C = \frac{H}{\tau} \text{ (бит/сек.)}, \qquad (3.18)$$

где τ - время передачи одного двоичного символа.

2. ***Пропускная способность*** (емкость канала связи) - максимальная скорость передачи информации по данному каналу связи, т.е. пропускная способность характеризуется максимальной энтропией для данного алфавита (первичного)

$$П = C_{max} = \frac{1}{\tau} \cdot H_{max} \text{ (бит/сек.)} \qquad (3.19)$$

Это в отсутствии помех. При наличии помех пропускная способность канала связи вычисляется как произведение количества принятых в секунду знаков n на разность энтропий источника сообщений и источника сообщений относительно принятого сигнала (условная энтропия)

$$П = n \cdot \left[H(A) - H\left(\frac{A}{B}\right)\right] \text{ (бит/сек.)} \qquad (3.20)$$

Скорость передачи бит за одну секунду называется 1 Бодом.

3. ***Избыточность сообщений.***

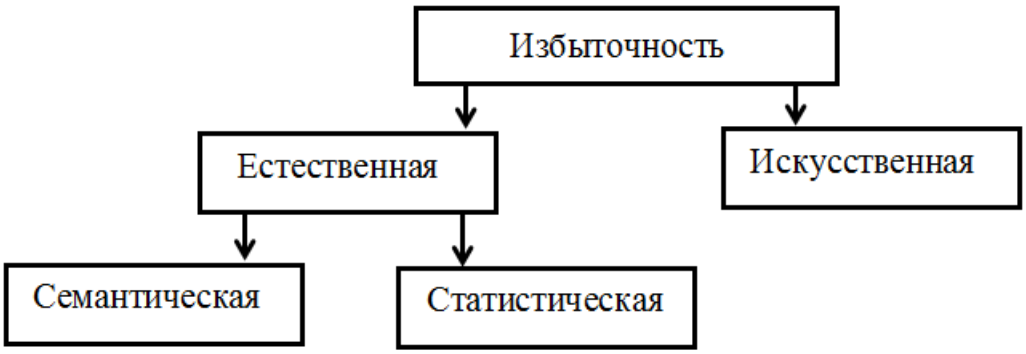


Рис 3.1 Классификация избыточности сообщений

Естественная избыточность присуща любому языку и служит для надежности процесса коммуникации. Она может быть семантической и статистической.

Семантическая избыточность заключается в том, что сообщение может быть выражено короче (телеграмма)

Статистическая избыточность обуславливается либо неравномерным распределением символов в сообщении, либо статистической связью между символами.

Если энтропия источника сообщений не равна максимальной для данного алфавита, то абсолютная недогруженность на символ сообщений такого источника определяется как:

$$\Delta D = (H_{max} - H) \text{ (бит/символ)}, \quad (3.21)$$

а избыточность определяется как

$$D = \frac{\Delta D}{H_{max}} = \frac{H_{max} - H}{H_{max}} = 1 - \mu, \quad (3.22)$$

где μ - коэффициент снятия.

H и H_{max} берутся для одного и того же алфавита.

4. ***Помехоустойчивость*** – способность канала связи (системы в целом) противостоять вредному действию помех. Различают атмосферные помехи, индустриальные помехи, импульсные помехи, и т.п. Увеличение помехоустойчивости связано с введением определенной избыточности, т.е. с увеличением объема передаваемой информации (объема сигнала). Если емкость канала связи это допускает, могут быть приняты меры, повышающие надежность передачи. Отметим некоторые из них:
   - Увеличение мощности сигнала;
   - Применение помехоустойчивого кодирования;
   - Применение помехоустойчивых методов модуляции сигнала;
   - Применение помехоустойчивых методов приема;
   - Применение каналов с обратной связью.

   Как правило, уменьшение избыточности увеличивает эффективность связи, но ухудшает ее помехоустойчивость.

Лекция 4.
Представление числовой информации в цифровых устройствах

Определение: Система счисления (СС) - совокупность приемов и правил для записи чисел цифровыми знаками.

Любая предназначенная для практического использования система счисления должна обеспечивать:

- возможность представления любого числа в рассматриваемом диапазоне величин;
- единственность представления;
- простоту оперирования с числами.

Самый простой способ записи чисел может быть описан выражением:

$$A_D = D_1 + D_2 + \cdots + D_k = \sum_{i=1}^{k} D_i, \qquad (4.1)$$

где A_D -запись числа A в системе счисления D, D_i -символы системы, образующие базу $D=\{D_1, D_2, \ldots\ldots D_k\}$. По этому принципу построены непозиционные системы счисления, для которых значение символа не зависит от его положения в числе. Принцип построения таких систем не сложен. Для их образования используют в основном операции сложения и вычитания. Классическим примером непозиционной системы счисления является Римская система счисления.

4.1 Позиционные системы счисления

В общем случае системы счисления можно построить по следующему принципу:

$$A_B = a_1B_1 + a_2B_2 + \ldots\ldots + a_nB_n, \qquad (4.2)$$

где A_B - запись числа A в системе счисления с основанием B_i, a_i - символ системы счисления с основанием B_i, B_i - ***база*** или ***основание*** системы счисления.

Если предположить, что $B_i = q^i$, то $B_i = q_iB_{i-1}$. Система счисления, удовлетворяющая равенству $B_i = q_iB_i$, называется ***позиционной*** системой счисления (ПСС). В качестве общего определения ПСС можно использовать следующее утверждение: ***В позиционной системе счисления значения символа числа зависит от его местоположения в числе***. Т.е. один и тот же знак (символ)принимает различное значение в зависимости от его положения в числе. Действительно, например, в числе 888 значение символа (цифры) “8” имеет разный вес в зависимости от его положения в записи числа: восемь, восемьдесят, восемьсот.

Естественная позиционная система счисления имеет место, если q -целое, положительное число. Любая позиционная система характеризуется основанием.

<u>Определение:</u> Основание (или базис) q естественной ПСС - есть количество символов, используемых для изображения числа в данной системе счисления.

Например, для 10-ной системы счисления имеем q=10 и имеем десять символов, принятых для обозначения чисел: 0,1,2,3,4,5,6,7,8,9. Вполне понятно, что возможно бесконечное множество позиционных систем счисления.

Для позиционной системы счисления справедливо равенство:

$$A_q = \sum_{i=-m}^{n} a_i q^i \qquad (4.3)$$

или $A_q = a_n q^n + a_{n-1} q^{n-1} + \ldots\ldots + a_1 q^1 + a_0 q^0 + a_{-1} q^{-1} + a_{-2} q^{-2} + \ldots\ldots + a_{-m} q^{-m}$

На практике используют сокращенную запись чисел:

$$a_n a_{n-1} a_{n-2} \ldots\ldots a_0 a_{-1} a_{-2} \ldots\ldots a_{-m}.$$

В таблице 4.1 приведены числа в десятичной системе счисления и их эквиваленты в других системах счисления.

Таблица 4.1

q=10	Эквиваленты десятичных чисел в других системах счисления					
	q=2	q=3	q=4	q=5	q=8	q=16
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	10	2	2	2	2	2
3	11	10	3	3	3	3
4	100	11	10	4	4	4
5	101	12	11	10	5	5
6	110	20	12	11	6	6
7	111	21	13	12	7	7
8	1000	22	20	13	10	8
9	1001	100	21	14	11	9
10	1010	101	22	20	12	A
11	1011	102	23	21	13	B
12	1100	110	30	22	14	C
13	1101	111	31	23	15	D
14	1110	112	32	24	16	E
15	1111	120	33	30	17	F
16	10000	121	100	31	20	10
17	10001	122	101	32	21	11
18	10010	200	102	33	22	12
19	10011	201	103	34	23	13

20	10100	202	110	40	24	14
21	10101	210	111	41	25	15
22	10110	211	112	42	26	16
23	10111	212	113	43	27	17
24	11000	220	120	44	30	18
25	11001	221	121	100	31	19
26	11010	222	122	101	32	1A

Для любой ПСС справедливо, что основание изображается числом “10” в своей системе, т.е. любое число можно записать в виде:

$$A_q=a_n\cdot 10^n + a_{n-1}\cdot 10^{n-1} + a_{n-2}\cdot 10^{n-2} + \ldots + a_1\cdot 10^1 + a_0\cdot 10^0 + $$
$$+ a_{-1}\cdot 10^{-1} + a_{-2}\cdot 10^{-2} + \ldots + a_{-m}\cdot 10^{-m} \qquad (4.4)$$

Вес разряда p_i числа в ПСС выражается соотношением:

$$p_i=q^i/q^0=q^i , \qquad (4.5)$$

где i -номер разряда.

Если разряд имеет вес $p_i=10^i$, то следующий разряд (старший) имеет вес $p_{i+1}=10^{i+1}$, а соседний младший - $p_{i-1}=10^{i-1}$. Такая взаимосвязь разрядов приводит к необходимости передачи информации между ними. Если в данном разряде накопилось значение единиц равное или больше q, то должна происходить передача единицы в соседний старший разряд (перенос).

Количество позиций (разрядов) в записи числа называется **длиной числа.** Для разных систем счисления характерна разная длина разрядной сетки, необходимая для записи одного и того же числа. Например, $100_{10}=64_{16}=144_8=1210_4=10201_3= 1100100_2$. Очевидно, что одно и тоже число в разных ПСС будет иметь меньшую длину для систем с большим основанием. В техническом аспекте длина числа интерпретируется как **длина разрядной сетки**. Если длина разрядной сетки задана, то это ограничивает максимальное по абсолютному значению число, которое может быть записано в данной разрядной сетке. Если длина разрядной сетки равна n, то

$$A_{q\,max}=q^n-1; \quad A_{q\,min}=-(q^n-1) \qquad (4.6)$$

<u>Определение:</u> Интервал представления чисел, заключенный между $A_{q\ max}$ и $A_{q\ min}$ называется диапазоном представления чисел (ДП).

$$A_{q\,\min} \le \text{ДП} \le A_{q\ max}$$

Правильный выбор системы счисления - важный практический вопрос. При выборе системы счисления необходимо учитывать:

- основание системы счисления определяет количество устойчивых состояний, которое должен иметь функциональный элемент, выбранный для представления разрядов числа;
- длина числа существенно зависит от основания системы счисления;
- система счисления должна обеспечивать простые алгоритмы выполнения арифметических и логических операций.

Для оценки эффективности использования ПСС введем показатель экономичности системы

$$C=q\cdot N,$$

где q - основание ПСС, N - длина разрядной сетки.

Если принять, что каждый разряд числа представлен не одним элементом с q устойчивыми состояниями, а q элементами, каждый из которых имеет одно устойчивое состояние, то показатель экономичности укажет условное количество оборудования, которое необходимо затратить на представление чисел в этой системе.

Итак, $A_{q\,max}=q^n-1$.

Нужно найти длину разрядной сетки.

$$A_{q\,max}+1=q^n,$$

откуда

$$n=\log_q(A_{q\,max}+1).$$

Тогда для любой системы счисления

$$C=q\cdot N=q\cdot \log_q(A_{q\,max}+1).$$

Для сравнения СС введем относительный показатель экономичности:

$$F=\frac{q\log_q(A_{q\,max}+1)}{2\log_2(A_{2\,max}+1)},$$

позволяющий сравнить любую систему счисления с двоичной. В таблице 4.2 приведены расчетные значения относительного показателя экономичности ПСС. График функции $F(q)$ приведен на рисунке 4.1

Таблица 3.2

q	***F***
2	1.0
3	0.946
4	1.0
6	1.148
8	1.333
10	1.505

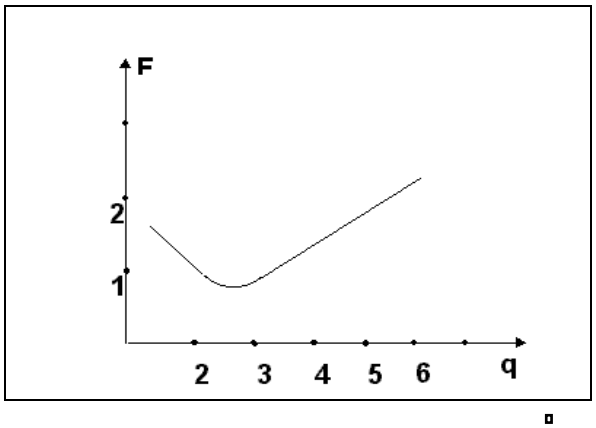


Рис. 4.1

Нижняя точка соответствует минимуму функции, определяемому из условия $\partial F/\partial q = 0$, что соответствует значению $q=e$. Следовательно, с точки зрения минимальных затрат условного оборудования, наиболее экономичной является система счисления с основанием e=2.728...

4.2 Перевод числовой информации из одной ПСС в другую

В соответствии с определением, числа в разных ПСС можно представить следующим образом:

$$A_{q_1} = \sum_{i=-m}^{n} a_i q_1^i = \sum_j^k b_j q_2^j = A_{q_2} \qquad (4.7)$$

Т.е. в общем виде задачу перевода из системы счисления с основанием q_1 в систему с основанием q_2 можно представить как задачу определения коэффициентов (символов) b_j нового ряда, изображающих это число в системе q_2.

Решать эту задачу можно и подбором коэффициентов b_j. Но основная трудность - выбор максимальной степени. Необходимо также иметь в виду, что в обеих системах должны выполняться условия:

$$0 \le a_i \le q_1 - 1 \quad \text{и} \quad 0 \le b_j \le q_2 - 1.$$

4.2.1 Перевод целых чисел делением на основание q_2 новой системы счисления

Целое число A_{q1} в системе с основанием q_2 записывается в виде:

$$A_{q_2} = b_k q_2^k + b_{k-1} q_2^{k-1} + \cdots + b_1 q_2^1 + b_0 q_2^0.$$

Переписав это выражение по схеме Горнера, получим:

$$A_{q_2} = \left(\cdots\left((b_k q_2 + b_{k-1})q_2 + b_{k-2}\right)q_2 + \cdots b_1\right)q_2 + b_0. \qquad (4.8)$$

Разделив выражение на величину основания q_2, получим целую часть

$$\left(\cdots\left((b_k q_2 + b_{k-1})q_2 + b_{k-2}\right)q_2 + \cdots b_1\right)$$

и первый остаток b_0. Разделив целую часть на q_2, получим второй остаток b_1. Повторяя процесс деления k+1 раз получим последнее частное b_k, которое по условию меньше основания системы q_2 и является старшей цифрой числа.

Например, переведем десятичное число 100 в позиционные системы счисления с основаниями q_2=2, 3, 4, 5, 6, 7, 8, 9, 16:

q_2=2

100	2
100	50
0	50
	0

50	2
50	25
0	24
	1

25	2
24	12
1	12
	0

12	2
12	6
0	6
	0

6	2
6	3
0	2
	1

3	2
2	1
1	

q_2=3

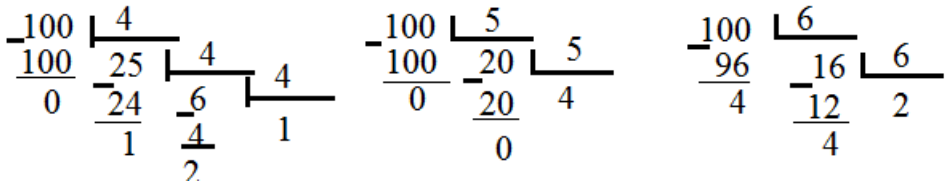
100	3
99	33
1	33
	0

33	3
33	11
0	9
	2

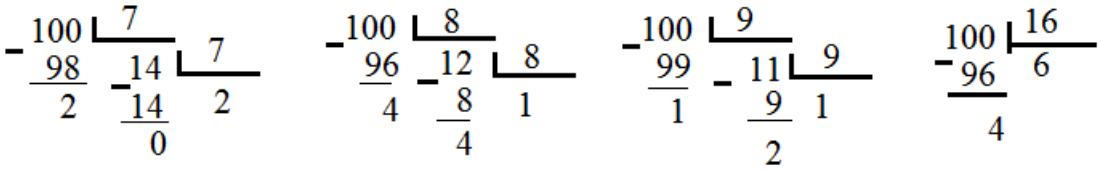
11	3
9	3
2	3
	0

3	3
3	1
0	

q_2=4 q_2=5 q_2=6



q_2=7 q_2=8 q_2=9 q_2=16



В результате перевода получили:

$100_{10}=1100100_2=10201_3=1210_4=400_5=244_6=202_7=144_8=121_9=64_{16}$

4.2.2 Перевод правильных дробей умножением на основание q_2 новой системы счисления

Пусть исходное число, записанное в системе с основанием q_1:

$$A_{q_1} = a_{-1}q_1^{-1} + a_{-2}q_1^{-2} + \cdots + a_{-m}q_1^{-m}$$

Тогда в новой системе данное число запишется в виде:

$$A_{q_2} = b_{-1}q_2^{-1} + b_{-2}q_2^{-2} + \cdots b_{-s}q_2^{-s}.$$

Переписав это выражение по схеме Горнера, получим:

$$A_{q_2} = q_2^{-1}(b_{-1} + q_2^{-1}(b_{-2} + \cdots + q_2^{-1}b_{-s})\cdots) \qquad (4.9)$$

Если правую часть выражения умножить на q_2, то получится новая неправильная дробь, в целой части которой будет число b_{-1}. Умножив затем оставшуюся дробную часть на величину основания q_2, получим дробь, в целой части которой будет b_{-2}, и т.д. Повторяя процесс умножения s раз, находим все s символов в новой системе счисления. При этом все действия должны выполняться по правилам q_1 арифметики и, следовательно, в целой части получающихся дробей будут проявляться эквиваленты цифр новой системы счисления. Переведем десятичную дробь 0.125 в системы счисления с основаниями, например, q_2=2, 3, 4, 5, 6:

q_2=2	q_2=3	q_2=4	q_2=5	q_2=6
0.125	0.125	0.125	0.125	0.125
× 2	× 3	× 4	× 5	× 6
0 25	0 375	0 5	0 625	0 750
× 2	× 3	× 4	× 5	× 6
0 5	1 125	2 0	3 125	4 500
× 2	× 3		× 5	× 6

1	0	0	375	0	625	3	000
		×	3	×	5		
		1	125	3	125		

В результате получили следующие результаты:

$0.125_{10}=0.001_2=0.010101_3=0.02_4=0.030303_5=0.043_6$

Для перевода неправильных дробей из одной ПСС в другую необходим раздельный перевод целой и дробной частей по вышеописанным правилам. Например, число из пятеричной системы необходимо перевести в шестеричную систему, т.е. имеем q_1=5, q_2=6, A_{q_1}=431.124_5. Сначала переводим только целую часть числа 431_5 делением на 6, которое в пятеричной системе счисления записывается как 11, т.е. $11_5=6_{10} = 10_6$. Все вычисления делаются по законам пятеричной арифметики:

431	11	11
33	34	3
101	33	
44	1	
2		

Т.е. получили, что $431_5=312_6$. Затем переводим дробную часть числа 0.124_5 умножением на 11, выполняя все действия по законам пятеричной арифметики:

0.	124
×	11
1	414
×	11
10	104
×	11
1	144
×	11
2	134

и т.д.

Отметим, что в результате умножения 0.414×11 получен результат 10.104 (т.е. 0.414×11=10.104), где пятеричное целое число 10_5 записывается уже в символике шестеричной системы счисления, т.е. как 5_6. (Напомним, что в любой позиционной системе счисления, число, равное основанию системы счисления записывается как 10. Поэтому $10_5=5_6$). Итак, $0.124_5=0.1512_6$.

Полученные значения соединяем “десятичной” точкой и получаем результат: $431.124_5=312.1512_6$.

4.2.3 Табличный метод перевода

В самом простейшем виде табличный метод перевода заключается в том, что имеется таблица всех чисел одной ПСС с соответствующими эквивалентами из другой ПСС, и перевод числа заключается в нахождении по таблице соответствующего эквивалента числа в нужной системе счисления. Использование подобных таблиц эквивалентности чисел нецелесообразно, т.к. они громоздки, требуют большой объем памяти для хранения, неудобны для использования.

Другой вид табличного метода перевода чисел из одной ПСС в другую заключается в том, что имеются таблицы эквивалентов в каждой системе только для цифр этих систем и степеней основания. Задача сводится к тому, что в выражении $A_q = \sum_{i=-m}^{i=n} a_i q^i$ для исходной системы счисления надо подставить эквиваленты из новой системы для всех цифр и степеней основания и произвести соответствующие действия по правилам q_2 арифметики. Полученный результат будет изображать число в новой (q_2) системе счисления.

Например, вышеописанным методом переведем десятичное число 123_{10} в двоичную систему счисления:

Десятичное число	Двоичный эквивалент десятичного числа
1	1
2	10
3	11
10^1	1010
10^2	1100100

Подставляя в десятичное число 123 его двоичные эквиваленты цифр и весов разрядов, и выполнив действия по законам двоичной арифметики, получаем:

$123_{10} = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 = 1 \cdot 1100100 + 10 \cdot 1010 + 11 \cdot 1 = 1100100 + 10100 + 11 = 1111011_2$.

Действительно, если перевести 123_{10} в двоичную систему методом деления получается тот же самый результат.

Пример 4.1 Табличным методом перевести число 431.124_5 в шестеричную систему счисления.

Для вычисления нам будет необходимо знать значения символов и весов разрядов пятеричной системы и их эквивалентные значения в

представлении шестеричной системы счисления. Пусть имеется такая таблица эквивалентов:

Символы и веса разрядов пятеричной системы (q=5)	Значения символов и весов разрядов для q=5 в шестеричной системе счисления
0	0
1	1
2	2
3	3
4	4
10^0	1
10^1	5
10^2	41
10^3	325
10^{-1}	0.111111…11
10^{-2}	0.0123501…
10^{-3}	0.001433…

$$431.124_5=4\cdot41+3\cdot5+1\cdot1+1\cdot0.11111+2\cdot0.01235+4\cdot0.001433=$$
$$244+23+1+0.11111+0.02514+0.01050=312.15115_6.$$

Как видно, результат получился тот же самый, что и при переводе по схеме Горнера.

Сразу отметим, что табличный метод очень удобен при переводе чисел из различных позиционных систем счисления в десятичную систему счисления, т.к. нужно выполнять операции в десятичной арифметике, которые нам более привычны.

4.2.4 Использование промежуточной системы счисления

Этот метод применяют для перевода из десятичной системы в двоичную и наоборот. В качестве промежуточной системы используется шестнадцатеричная или восьмеричная системы счисления. Суть метода заключается в том, что десятичное число переводят, например, в шестнадцатеричную систему счисления, а затем каждый символ полученного числа в шестнадцатеричной системе счисления заменяют четырехразрядным (тетрадой) двоичным эквивалентом. Это следует из того, что шестнадцатеричная система счисления связана с двоичной системой счисления соотношением $16^n=(2^4)^n$. Аналогично для перевода можно использовать восьмеричную систему счисления, но при этом в полученном выражении каждый символ числа заменяется трехразрядным (триадой) двоичным эквивалентом, т.к. восьмеричная система счисления

связана с двоичной соотношением: $8^n=(2^3)^n$. В качестве промежуточной системы можно использовать так же и четверичную систему счисления.

Пример 4.2. Число 12345_{10} перевести в двоичную систему счисления.
Решение.

1. Данное число переводим в 16-ричную систему: $12345_{10}=3039_{16}$.
2. Каждый символ полученного числа заменяем двоичной тетрадой и получаем искомый результат: 11000000111001_2.

4.3 Формы представления чисел

Любое число имеет многообразие форм своего представления. Действительно, число 0.02345 можно записать так: $2345\cdot10^{-5}$, или $2.345\cdot10^{-2}$, или $0.2345\cdot10^{-1}$ и т.д. Разнообразие форм в записи одного числа может послужить причиной затруднений для работы цифрового устройства. Во избежание этого нужно либо создать специальные алгоритмы распознавания числа, либо указывать каждый раз форму его записи, что является более просто.

Существует две формы записи чисел: естественная и нормальная.

При естественной форме число записывается в естественном натуральном виде, в соответствии со смысловым значением: 123 - целое число, 0.02345 - правильная дробь, 12.345 - неправильная дробь.

При нормальной форме запись одного числа может принимать разный вид, в зависимости от ограничений, накладываемых на ее форму. Например, 123= $=1.23\cdot10^2= 0.123\cdot10^3=0.123000\cdot10^{-3}$ и т.д.

Определение: Автоматное (машинное) изображения числа-представление числа А в разрядной сетке цифрового автомата.

Условно автоматное изображение числа A обозначается $[A]$. Справедливо соотношение: $A=[A]\cdot K_A$, где K_A - коэффициент, величина которого зависит от формы представления числа в автомате.

4.3.2 Представление чисел в формате с фиксированной (запятой) точкой

Естественная форма представления числа в ЭВМ (или ином цифровом автомате) характеризуется тем, что положение его разрядов в автоматном изображении остается всегда постоянным, независимо от самого числа. Т.е. длина разрядной сетки строго фиксирована, а так же фиксирована ДРС целой и дробной частей. В связи с этим существует так же и другое название этой формы представления чисел - ***представление чисел в форме с фиксированной запятой (точкой)***. В ЭВМ эта форма

используется преимущественно для представления целых чисел. Так как числа бывают положительные и отрицательные, то в разрядной сетке при их машинном представлении один разряд отводится под знак числа, а остальные образуют поле числа. Знаковый разряд может располагаться как в начале, так и в конце числа. Принято, что “0” - соответствует положительному, а “1” - отрицательному числам.



Если поле числа включает n разрядов, то диапазон представления целых чисел в этом случае ограничивается значениями $-(2^n-1) \div +(2^n-1)$. Для приведенного здесь примера имеем n=11, и тогда диапазон представления целых чисел: -2047 ÷ +2047.

При записи на бумаге принято целую часть числа от дробной отделять запятой (в США - точкой). Также первоначально стали представлять числа и в ЭВМ, но оказалось, что в достаточно длинном 32-разрядном машинном слове помещается число, которое после перевода в десятичную систему счисления содержит не более десяти цифр:

$$\pm(2^{31}-1) = \pm 2\ 147\ 483\ 647.$$

При инженерных и научных расчётах часто требуется более трёх знаков после запятой. Если под дробную часть отвести четыре разряда, т. е. зафиксировать запятую перед четвёртым разрядом справа, то самое большое по модулю число будет меньше 220 000:

214748.3647 .

Этого совершенно недостаточно для практических расчётов, поэтому для представления действительных чисел используют форму представления чисел с плавающей точкой.

Целые числа в программах, в основном, используются для нумерации, индексации и диапазона

-2 147 483 647 ÷ +2 147 483 647,

что вполне хватает.

Итак, под числами с фиксированной запятой понимают целые числа. На практике применяют 8-ми, 16-и, 32-х и 64-х разрядные форматы (со знаком или без знака) числа с фиксированной запятой. Пример записи числа $-500_{10} = 111110100_2$ в 16-битном формате со знаком

1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0

Крайний левый разряд отведён под знак числа: 0 - плюс, 1 - минус.

Для того, чтобы определить, сколько двоичных разрядов будет занимать целое положительное число N после перевода из десятичной системы счисления в двоичную, нужно вычислить $k = log_2 N$ и взять ближайшее большее, чем k, целое число.

Примеры

$\log_2 15 = 3.90$ $k = 4$ $15_{10} = 1111_2$

$\log_2 16 = 4$ $k = 5$ $16_{10} = 1\ 0000_2$

$\log_2 500 = 8{,}97$ $k = 9$ $500_{10} = 1\ 1111\ 0100_2$

$\log_2 1000 = 9.96$ $k = 10$ $1000_{10} = 11\ 1110\ 1000_2$

$\log_2 1024 = 10$ $k = 11$ $1024_{10} = 100\ 0000\ 0000_2$

4.3.2 Представление чисел в форме с плавающей запятой (точкой)

Как известно, для записи очень больших, или очень маленьких по модулю чисел используют полулогарифмическую (нормальную) форму представления чисел. Например, расстояние от Земли до Солнца удобнее записывать не в виде 149 600 000 км, а в виде $1.496 \cdot 10^8$ км, или даже с меньшей точностью в виде $1.5 \cdot 10^8$ км.

Многие физические величины практически можно записать только в нормальной форме. Например, массу электрона m_e:

$$m_e = 9.1091 \cdot 10^{-31} \text{ кг.}$$

В нормальной форме число A запишется как

$$A_н = m_A \cdot p_{A,} = m_A \cdot q^p , \qquad (4.10)$$

где m_A - мантисса числа A, q – основание, p – порядок, q^p характеристика числа A.

Как видно из ранее изложенного, такое представление чисел не однозначно; для определения обычно вводят некоторые ограничения. Наиболее распространено для представления ограничения вида:

$$q^{-1} \le |m_A| < 1, \qquad (4.11)$$

где q - основание системы счисления.

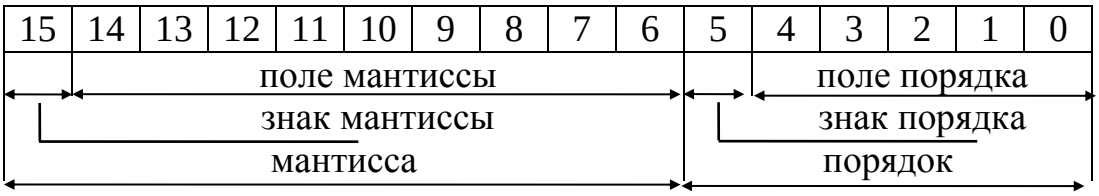
Применительно к ЭВМ условие (4.11) часто записывают в виде:

$$0.5 \le |m_A| < 1$$

Определение: Форма представления чисел, для которой справедливо условие $q^{-1} \le |m_A| < 1$, называется нормализованной формой представления числа.

Поскольку в этом случае абсолютное значение мантиссы лежит в пределах от q^{-1} до $1\text{-}q^{-n}$, где n - количество разрядов для изображения мантиссы без знака, положение разрядов числа в его автоматном представлении не постоянно. Поэтому такую форму представления

называют **формой представления с плавающей точкой**. Формат машинного изображения числа с плавающей точкой должен содержать знаковые части и поля для мантиссы и порядка. Выделяются специальные разряды для изображения знака числа (мантиссы) и знака порядка. Например, для 16-разрядной сетки условно формат можно представить в виде:



Рассмотрим пример записи чисел в формате с плавающей точкой.

Пусть в разрядной сетке необходимо записать двоичные числа:

1. $A_1 = -101100.1111_2$
2. $A_2 = 0.000110010111_2$

Прежде всего запишем эти числа в нормальной форме. Порядок чисел выбираем таким образом, чтобы для них выполнялось условие 4.11. Тогда получаем:

$A_1 = -0.101101111 \cdot 10^{110}$;

$A_2 = 0.110010111 \cdot 10^{-11}$.

Поскольку для изображения выделено, например, пять разрядов и один для знака, то их машинные изображения и машинные изображения мантисс будут иметь вид:

$[p_{A_1}] = 000110$ $[p_{A_2}] = 100011$

$[m_{A_1}] = 1.101101111$ $[m_{A_2}] = 0.110010111$

Число A_1:

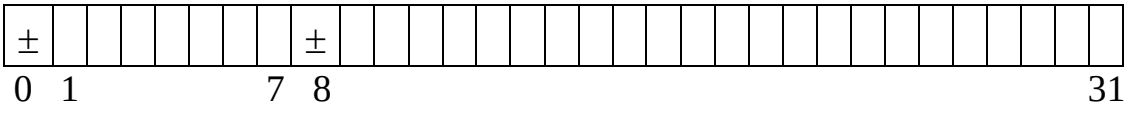
1	1	0	1	1	0	1	1	1	1	0	0	0	1	1	0

Число A_2:

0	1	1	0	0	1	0	1	1	1	1	0	0	0	1	1

Часто формат с плавающей точкой представляют и в другом виде, при котором, например, сначала записывается порядок, а затем – мантисса.

Например, простейший 32-разрядный формат числа с плавающей запятой имеет вид:



В разряды 0 - 7 записывается порядок числа. В разряде 0 - знак порядка.

В разряды с 8-го по 31-й помещается мантисса. В 8-м разряде - знак мантиссы.

Представим, например, в форме с плавающей точкой число $A = 12.5_{10} = 1100.1_2$. В нормализованном виде

$$A = ((12.5/2^4)\cdot 2^4)_{10} = (0.78125\cdot 2^4)_{10} = (0.11001\cdot 10^{100})_2$$

(Конечно, это число можно перевести в 2-ю по схеме Горнера и далее записать его в нормализованном виде по правилам двоичной арифметики: $A=(1100.1=0.11001\cdot 10^{100})_2$).

Тогда в 32-разрядной ячейке памяти число A выглядит так:

0	0000100	0	11001000000000000000000
+	4	+	0.78125

Большинство людей за всю свою жизнь ни разу не сталкивалось с числом, состоящим из такого большого количества цифр, что для его записи не хватало одной строки листа бумаги. К сожалению, в ЭВМ числа часто не помещаются в заданный формат.

Самое большое число с плавающей точкой определяется характеристикой $A = q^p$.

При $q = 2$ и $p_{\max} = 2^7-1 = 127$ получаем $A_{max}=2^{127}=10^{38}$.

Данная оценка удобна, но не совсем точна. Запишем в рассмотренном выше 32-разрядном формате самое большое число:

0	1111111	0	11111111111111111111111

Переведём полученное число в десятичную систему счисления. Сначала переведём мантиссу:

$$0.111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = 1 - 2^{-23}.$$

Умножив мантиссу на характеристику, получим в десятичной системе счисления самое большое число, которое можно записать в 32-разрядном формате с плавающей точкой:

$$(1 - 2^{-23})\cdot 2^{127} = 1{,}70141\cdot 10^{38}$$

Самое маленькое положительное число, которое можно записать в рассматриваемом формате, отличается по форме только знаковым разрядом порядка и в десятичной системе счисления оно имеет вид:

$$(1 - 2^{-23})\cdot 2^{-127} = 5{,}87747\cdot 10^{-39}.$$

Итак, модуль числа A в рассматриваемом формате должен находиться в диапазоне:

$$5{,}87747\cdot 10^{-39} \le |A| \le 1{,}70141\cdot 10^{38}$$

4.3.3 Представление отрицательных чисел

Основной операцией двоичного сумматора является операция сложения. Операцию арифметического вычитания заменяют операцией алгебраического сложения: $A-B = A+(-B)$

Для машинного представления отрицательных чисел используют коды: ***прямой, дополнительный, обратный.***

Прямой код числа $A=-0.a_1a_2.....a_n$- есть машинное изображение этого числа в виде $[A]_{пр}=1a_1a_2....a_n$. В прямом коде все цифровые разряды отрицательного числа остаются неизменными, а в знаковой части записывается единица. Положительное число в прямом коде не меняет своего изображения. Например: если A=-0.101110 , то $[A]_{пр}$=1101110, т.е. в знаковой части записывается единица; если A= 0.110111, то $[A]_{пр}$=0110111, т.е. положительное число не меняет своего изображения.

В прямом коде в разрядную сетку можно записать следующее максимальное по абсолютному значению число: $A_{пр\ мах}$=01111....1111=2^n-1, где n - количество разрядов разрядной сетки. Диапазон изменения машинных изображений для прямого кода составит: $-(2^n-1) \le [A]_{пр} \le (2^n-1)$.

Если положить, что для представления чисел используется формат с фиксированной точкой, и при этом числа представляются в виде правильной дроби, т.е. $-1<[A]_ф<1$, то в этом случае максимальное по абсолютному значению число в прямом коде запишется как $A_{пр\ мах}$ =0.1111...1111=1-2^n , где n -количество разрядов разрядной сетки цифрового автомата. И тогда правила преобразования чисел в прямой код можно сформулировать в виде:

$$[A]_{\text{пр}} = \begin{cases} A, \text{если } A \geq 0, \\ 1 + |A|, \text{если } A < 0 \end{cases} \qquad (4.12)$$

Обратный код числа $A=-0a_1a_2....a_n$ - есть такое машинное изображение этого числа $[A]_{\text{об}} = 1\bar{a}_1\bar{a}_2 \cdots \bar{a}_n$, для которого $\bar{a}_i = 0$, если $a_i = 1$, и $\bar{a}_i = 1$, если $a_i = 0$. Из определения следует, что обратный код двоичного числа является инверсным изображением самого числа, в котором все разряды исходного числа принимают инверсное (обратное) значение, т.е. все нули заменяются на единицы, а единицы на нули. Например, если A=-01001100111, то $[A]_{об}$=1 0110011000 .

Для обратного кода чисел, представленных в форме с запятой, фиксированной перед старшим разрядом (т.е. – правильная дробь), справедливо соотношение:

$$|A| + [A]_{\text{об}} = q - q^{-n}, \qquad (4.13)$$

где $|A|$ - абсолютная величина A, n - количество разрядов после запятой в изображении числа.

Правила преобразования чисел в обратный код можно сформулировать следующим образом:

$$[A]_{об} = \begin{cases} A, \text{если } A \geq 0, \\ q - q^{-n} - |A|, \text{если } A < 0. \end{cases} \quad (4.14)$$

В обратном коде можно изображать максимальное положительное число $[A]_{об\ мах}$ =0.111...111=(1-2^{-n}) и наибольшее отрицательное число $[A]_{об\ min}$ = - 0.111...111= -(1-2^{-n}). Следует иметь в виду неоднозначное изображение нуля в обратном коде: +0 изображается 0.000...000, то -0 изображается 1.111...111.

Если необходимо преобразовать в обратный код целое число, то это можно выполнить в соответствии с формулой (4.15), которая получается из (4.14).

$$[A]_{об} = \begin{cases} A, \text{если } A \geq 0, \\ 2^n - 1 - |A|, \text{если } A < 0, \end{cases} \quad (4.15)$$

где n – количество разрядов в формате числа A (в n входит и знаковый разряд), или проинвертировать все числовые разряды слова, т.е. заменить все единицы в двоичном коде на нули, а нули на единицы.

Например, для представления числа -100$_{10}$=-1100100$_2$ в 10-разрядной сетке двоичного кода имеем:

$$2^{10}\text{-1-100=1024-1-100=923=39}B_{16} = 1\ 110011011.$$

Конечно, намного проще получить обратный код двоичного числа в заданной разрядной сетке инвертированием его прямого кода (только цифровой части). $[A]_{пр}$ = 1 001100100. Тогда $[A]_{об}$=1 110011011.

Дополнительный код числа *A=-0a$_1$a$_2$....a$_n$* -есть такое машинное изображение этого числа $[A]_д = 1\bar{a}_1\bar{a}_2 \cdots \bar{a}_n$, которого $\bar{a}_i = 0$, если $a_i = 1$, и $\bar{a}_i = 1$, если $a_i = 0$, за исключением последнего значащего разряда, для которого $\bar{a}_n = 1$ при $a_n = 1$.

Для дополнительного кода чисел, представленных в форме с запятой, фиксированной перед старшим разрядом (т.е. – правильная дробь), справедливо соотношение:

$$|A| + [A]_д = q. \quad (4.16)$$

Т.е., дополнительный код числа является математическим дополнением основанию системы счисления.

Т.к. положительные числа не меняют своего изображения в дополнительном коде, то правила преобразования в дополнительный код можно записать следующим образом:

$$[A]_д = \begin{cases} A, \text{если } A \geq 0, \\ q - |A|, \text{если } A < 0. \end{cases} \quad (4.17)$$

Сравнивая 4.16 и 4.17 видно, что

$$[A]_д = [A]_{об} + q^{-n}.$$

Это выражение используют для получения дополнительного кода отрицательного числа следующим образом:

Сначала инвертируется цифровая часть исходного кода, в результате чего получается его обратный код, затем добавляется единица в младший разряд цифровой части числа и тем самым получается дополнительный кода исходного изображения.

Пример 4.3. Найти обратный и дополнительный код числа A= - 0.11011100110.

Решение. Используя определение обратного кода, получаем $[A]_{об}$ =100100011001.

Для нахождения дополнительного кода прибавим единицу к младшему разряду полученного изображения:

$$\begin{array}{r} 100100011001 \\ + \qquad\qquad 1 \\ \hline [A]_д = 100100011010 \end{array}$$

Ответ: $[A]_{об}$ = 100100011001; $[A]_д$ = 100100011010

Для получения двоичного дополнительного кода целого отрицательного используется формула:

$$[A]_д = [A]_{об} + 1 = 2^n - |A|. \qquad (4.18)$$

Например, для представления числа -100_{10} = -1100100_2 в дополнительном коде в 10-разрядной сетке в соответствии с (4.18) получаем:

$[A]_д = 2^{10}$-100=1024-100=924=39C_{16} = 1 110011100

Использование различных способов изображения отрицательных чисел в цифровом устройстве (например, ЭВМ) обуславливает целый ряд особенностей выполнения операции алгебраического сложения двоичных чисел.

4.3.4 Погрешности представления чисел

Представление цифровой информации в ЭВМ, как правило, влечет за собой появление погрешностей (ошибок), величина которых зависит от формы представления чисел и от длины разрядной сетки цифрового автомата.

Абсолютная погрешность представления – разность между истинным значением входной величины А и ее значением, полученным из машинного изображения A_M, т.е. $\Delta[A]=A-A_M$.

Относительная погрешность представления – величина, определяемая соотношением:

$$\delta[A] = \frac{\Delta[A]}{A_M} \qquad (4.19)$$

Пример 4.4. Часто принимают значение π равным 3.14. Однако эта величина может быть получена и с большей точностью. Если принять, что точное значение π=3.141159265, то абсолютная погрешность будет равна в данном случае $\Delta[\pi]$=0.00159265, а относительная погрешность - $5.072133 \cdot 10^{-4}$.

Часто некоторая величина в одной системе счисления имеет конечное значение, а в другой системе счисления становится бесконечной дробью. Например, число 0.1_{10} имеет конечное десятичное представление, но в двоичной системе это число выражается бесконечной (в данном случае периодической) дробью: $0.0001100110011\ldots_2$. Следовательно, при переводе чисел из одной системы счисления в другую неизбежно возникают погрешности, оценить которые нетрудно, если известны истинные значения входных чисел.

Если учесть, что для машинного представления чисел справедливо соотношение: $A=[A]\cdot K_A$, где K_A – коэффициент, величина которого зависит от формы представления числа в цифровом автомате, и при этом масштабный коэффициент K_A выбирают так, чтобы абсолютное значение машинного изображения числа A в системе счисления с основанием q=2 было всегда меньше 1. Для представления чисел в ЭВМ в общем случае примем K_A=1. Также принимая во внимание ограниченность разрядной сетки для представления правильной дроби ($-1<[A]_Ф<1$, где $[A]_Ф$ – машинное изображение числа в формате с фиксированной точкой), то с учетом вышесказанного будем иметь:

$$A_q = K_A\left[a_{-1}q^{-1} + a_{-2}q^{-2} + \cdots + a_{-n}q^{-n} + a_{-(n+1)}q^{-(n+1)} + \cdots\right] = \\ = \sum_{i=-1}^{\infty} a_i q^i \qquad (4.20)$$

Т.к. длина разрядной сетки равна n разрядов после запятой, то абсолютная погрешность перевода десятичного числа в систему с основанием q будет равна:

$$\Delta[A] = a_{-(n+1)}q^{-(n+1)} + \cdots + a_{-(n+m)}q^{-(n+m)} + \cdots = \sum_{i=-(n+1)}^{\infty} a_i q^i. \quad (4.21)$$

Оценим погрешность представления чисел в двоичной системе счисления. Имеем q=2. Максимальное значение погрешности будет, если a_i=1, и тогда имеем:

$$\Delta[A]_{max} = \sum_{i=-(n+1)}^{\infty} 1 \cdot 2^i = 2^{-n}\sum_{i=-1}^{\infty} 2^i = 2^{-n} \qquad (4.22)$$

Т.е. максимальная погрешность перевода десятичного числа в двоичное представление не будет превышать единицы младшего разряда разрядной сетки. Вполне очевидно, что минимальная погрешность перевода будет равна нулю. Усредненная погрешность перевода чисел, равная среднему

арифметическому минимальной и максимальной погрешностей, будет равна:

$$\Delta[A] = \frac{0+2^{-n}}{2} = 0.5 \cdot 2^{-n}. \quad (4.23)$$

Для представления чисел в формате с фиксированной точкой абсолютное значение машинного изображения числа (для правильной дроби) равно:

$$2^{-n} \le \left|[A]_{\text{ф}}\right| \le 1 - 2^{-n}.$$

Следовательно, относительная погрешность представления минимального значения числа составит:

$$\delta[A]_{\text{ф}\,min} = \frac{\Delta[A]}{[A]_{\text{ф}\,max}} = \frac{0.5 \cdot 2^{-n}}{1-2^{-n}}. \quad (4.24)$$

Для ЭВМ, как правило, n=16÷64, поэтому $1 >> 2^{-n}$ и тогда получаем:

$$\delta[A]_{\text{ф}\,min} = \frac{\Delta[A]}{[A]_{\text{ф}\,max}} = \frac{0.5 \cdot 2^{-n}}{1} = 0.5 \cdot 2^{-n}. \quad (4.25)$$

Аналогично, для оценки максимального значения получаем:

$$\delta[A]_{\text{ф}\,max} = \frac{\Delta[A]}{[A]_{\text{ф}\,min}} = \frac{0.5 \cdot 2^{-n}}{2^{-n}} = 0.5. \quad (4.26)$$

Из последнего выражения видно, что погрешности представления малых чисел в формате с фиксированной точкой могут быть очень значительными. Поэтому формат с фиксированной точкой используется в ЭВМ только для представления целых чисел.

Для представления чисел в формате с плавающей точкой абсолютное значение мантиссы должно удовлетворять условию:

$$2^{-1} \le |[m_A]| \le 1 - 2^{-n}. \quad (4.27)$$

Погрешность $\Delta[A]_{max}=2^{-n}$ есть погрешность мантиссы. Для нахождения погрешности представления числа в формате с плавающей точкой величину этой погрешности нужно умножить на величину порядка числа p_A, и тогда получаем:

$$\delta[A]_{\text{П}\,max} = \frac{0.5 \cdot 2^{-n} p_A}{2^{-1} p_A} = 2^{-n}, \quad (4.28)$$

$$\delta[A]_{\text{П}\,min} = \frac{0.5 \cdot 2^{-n} p_A}{(1-2^{-n}) p_A} = 2^{-(n+1)}, \quad (4.29)$$

где n – количество разрядов для представления мантиссы числа. Из последних выражений следует, что погрешность представления чисел в формате с плавающей точкой не зависит от величины числа и определяется длиной разрядной сетки представления мантиссы числа.

Рассмотрим пример по оценке представления чисел в ЭВМ в формате с плавающей точкой.

Как известно из алгебры, в 10-ной системе счисления дробь, знаменатель которой не может быть приведён к виду 10^n, n = 1, 2, ..., не может быть представлена точно в виде десятичной дроби. Например,

$$\frac{7}{50} = \frac{14}{100} = \frac{14}{10^2} = 0.14;$$

$$\frac{5}{24} = \frac{5}{(3\cdot2\cdot2\cdot2\cdot2)} = 0.208333(3);$$

$$\frac{2}{3} = 0.666(6)$$

Аналогичным свойством обладает и двоичная система счисления.

Точно в формате с плавающей запятой представляется число, модуль которого можно записать в виде следующей несократимой дроби:

$\frac{A}{2^n}$, где A и n – целые положительные числа.

Может показаться неожиданным, что 0,1 (одна десятая) не может быть представлена точно в формате с плавающей запятой, а 0,25 – представима:

$$0.1_{10} = \frac{1}{(2\cdot5)} = 0.00011001100\ldots = 0.00011(0011)_2;$$

$$0.125_{10} = \frac{1}{8} = \frac{1}{2^3} = 0.001_2.$$

Количество двоичных знаков в числе x определяется количеством разрядов в мантиссе. В рассматриваемом нами 32-разрядном формате длина мантиссы, как мы определили ранее, составляет 23 двоичных разрядов. Тогда количество десятичных знаков

$$R = \log_{10} 2^{23} = 23 \cdot \log_{10} 2 \cong 6.92369 \cong 7.$$

Целое число N не имеет погрешности в формате с плавающей запятой, если в двоичном виде номер его самого младшего разряда, содержащего единицу, меньше или равен числу разрядов в мантиссе. Например, $N = 2^{33}$ представляется точно, а $N = 2^{33} - 1$, как показано в приведённом ниже примере, имеет погрешность.

Теория погрешностей вычислений в формате с плавающей запятой очень сложна. Приведем примеры возникновения таких погрешностей.

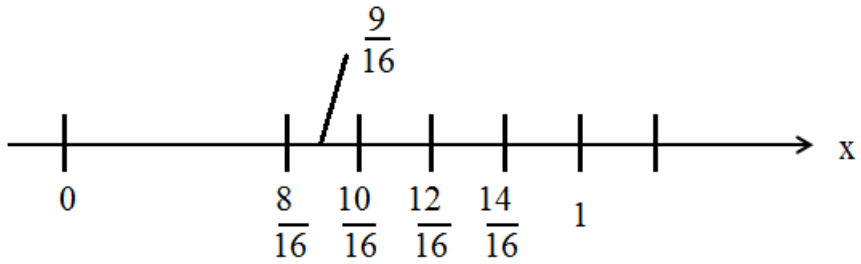
Для простоты рассмотрим семибитный формат числа с плавающей запятой:

±			±			
0	1	2	3	4	5	6

В этом формате при значении порядка $p = 0$ точно представимы только 4 числа: 1/2=4/8; 5/8; 6/8 и 7/8. В памяти ЭВМ эти числа хранятся так:

0	0	0	**0**	1	0	0
0	0	0	**0**	1	0	1
0	0	0	**0**	1	1	0
0	0	0	**0**	1	1	1

Разместим эти числа на числовой оси:



Все числа, заключённые между 1/2 и 5/8 в выбранном нами формате не могут быть представлены точно. Они будут округлены либо в меньшую сторону до 1/2, либо в большую до 5/8. Попытаемся представить в выбранном формате $9/16 = (0.1001)_2$:

0	0	0	**0**	1	0	0	1

Младший разряд вышел за пределы формата и вместо 9/16 в памяти ЭВМ будет храниться 8/16. В общем случае погрешность представления числа с плавающей запятой

$0 \le \Delta x < 2^{-m} \cdot 2^{p}$, где m-число разрядов в мантиссе, исключая знаковый.

На практике для представления чисел с плавающей запятой чаще всего используются машинные слова длиной либо 32 либо 64 бита. Запишем в 32-разрядное слово число $x = 2^{33} - 1$. В двоичной системе счисления - это число из 33-х единиц. Найдём мантиссу и порядок этого числа в нормализованном виде:

$x = \frac{2^{33}-1}{2^{33}} \cdot 2^{33} = 0.111\ 111\ 111\ 111\ 111\ 111\ 111\ 111\ 111\ 111 \cdot 10^{100001}$.

Для того чтобы, не теряя точности, разместить это число, требуется слово длиною 41 бит (7 бит под порядок и 34 бита под мантиссу).

Разместив это число в 32-битном формате с 24-разрядной мантиссой, считая знак, мы потеряем 10 значащих цифр исходного числа x:

0	0100001	0	11111111111111111111111	1 1 1 1 1 1 1 1 1 1
0	1 7	8	31	10 разрядов

23 разряда

Вычислим погрешность числа x в 32-разрядном двоичном формате с плавающей запятой:

$$\Delta x = x - m_x \cdot p_x = (2^{33} - 1) - (1 - 2^{-23}) \cdot 2^{33} = 2^{10} - 1 = 1023,$$

где m_x – 24-разрядная мантисса.

В десятичной системе счисления

$$2^{33} = 8\ 589\ 934\ 591;$$
$$(1- 2^{-23})\cdot 2^{33} = 8\ 589\ 933\ 568,$$

т.е. в форме с плавающей точкой точно представлены только 6 старших разрядов исходного числа. Все числа в диапазоне $8\ 589\ 933\ 568 \le x \le 8\ 589\ 934\ 591$ в 32-разрядном двоичном формате с плавающей запятой будут представлены одним числом - 8 589 933 568.

Можно оценить погрешность представления числа N в формате с плавающей запятой без перевода из десятичной системы счисления в двоичную. Для простоты рассмотрим пример, в котором мантисса в формате с плавающей запятой имеет семь двоичных разрядов, а N=1234. В двоичном представлении N имеет 11 разрядов:

$$]\log_2 1234[+ 1 = 11, \text{где }]x[- \text{целая часть числа } x.$$

Так как мантисса имеет семь разрядов, то из числа N при записи в ячейку будут отброшены 11 – 7 = 4 разряда. Далее

- разделив N на 2^4 получаем 1234/16 = 77.125;
- вычислим погрешность: $\Delta = 0{,}125\cdot 16 = 2$.

Проверим результат прямым переводом N в двоичную систему счисления:

$1234 = 1024 + 128+64+16+2= 10011010010_2$;
Мантисса $m = 0, 1001101_2$;
Порядок $p = 1011_2$.

0	1	0	1	1	**0**	1	0	0	1	1	0	1	0 0 1 0
+	1			4	+	6						12	остаток

Переведём полученное число с плавающей запятой в десятичную систему счисления:

$N_{пл}=0.1001101\cdot10^{1011}=10011010000=2^{10}+2^{7}+2^{6}+2^{4}=$

$=1024+128+64+16=1232.$

$\Delta=N-N_{пл}=1234-1232=2.$

Следует сразу отметить, что погрешности возникают и при выполнении арифметических операциях над числами в формате с плавающей точкой, что будет рассмотрено позже.

Лекция 5

5. Основы двоичной арифметики

5.1 Формальные правила двоичной арифметики

Простота выполнения арифметических действий в двоичной системе счисления очевидна:

Сложение	Вычитание	Умножение
0+0=0	0-0=0	0×0=0
0+1=1	1-0=1	0×1=0
1+0=1	1-1=0	1×0=0
1+1=(1)0	0-1=(1)1	1×1=1
(единица переноса в старший разряд)	(заем единицы в старшем разряде)	

В основе арифметическо-логического устройства любой ЭВМ лежит сумматор, для которого разработаны алгоритмы выполнения арифметических операций. В общем случае выполнение арифметических действий сумматором представляется простым выражением:

$$C=A \nabla B,$$

где ∇ - знак арифметического действия. Так как ЭВМ, как и любой другой цифровой автомат, оперирует только с машинным (автоматным) изображением чисел, то правильнее бы стоило записать:

$$[C]=[A]\nabla[B],$$

где [ ] –обозначено машинное представление операндов. (***Операнды*** – это числа, непосредственно участвующие в арифметической операции).

Рассмотрим формальные правила выполнения арифметических операций сложения на уровне разрядов операндов.

На основе правил двоичной арифметики можно записать правила сложения двоичных цифр так, как показано в таблице 5.1 и 5.2, где a_i, b_i – разряды операндов A и B соответственно, c_i – разряд суммы, n_i –перенос из данного разряда в соседний (старший) разряд.

Двоичный полусумматор – устройство, выполняющее арифметические действия в соответствии с таблицей 5.1

Как видно из таблицы 5.1 появление единицы переноса при сложении двух разрядов несколько изменяет правила сложения двоичных цифр. Обобщая вышеизложенное, можно сформулировать правила поразрядных действий при сложении операндов A и B:

$$a_i+b_i+n_{i-1}=c_i+n_i,$$

где n_{i-1} – перенос из (i-1) –го разряда, n_i – перенос в (i+1) разряд.

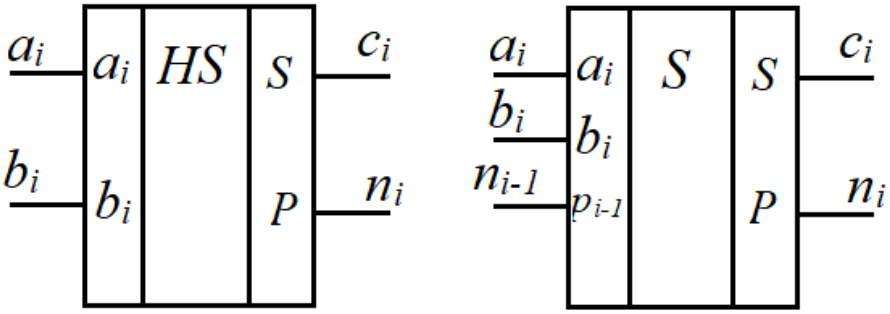
Таблица 5.1

a_i	b_i	c_i	n_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Таблица 5.2

a_i	b_i	n_{i-1}	c_i	n_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Двоичный сумматор – устройство, выполняющее арифметические действия в соответствии с таблицей 5.2. Условные обозначения полусумматора и сумматора приведены на рисунке 5.1а и 5.1б соответственно



а б

Рис. 5.1

5.2 Сложение чисел в форме с фиксированной запятой на двоичных сумматорах

5.2.1 Двоичный сумматор прямого кода (ДСПК)

Двоичный сумматор, в котором отсутствует цепь поразрядного переноса между старшим цифровым и знаковым разрядом, называется двоичным сумматором прямого кода – ДСПК. Условная схема ДСПК может быть представлена в виде, показанном на на рис. 5.2.

Как видно из таблицы 5.1 появление единицы переноса при сложении двух разрядов несколько изменяет правила сложения двоичных цифр. Обобщая вышеизложенное, можно сформулировать правила поразрядных действий при сложении операндов *A* и *B*:

$$a_i+b_i+n_{i-1}=c_i+n_{i,}$$

где n_{i-1} – перенос из (i-1) –го разряда, n_i – перенос в (i+1) разряд.

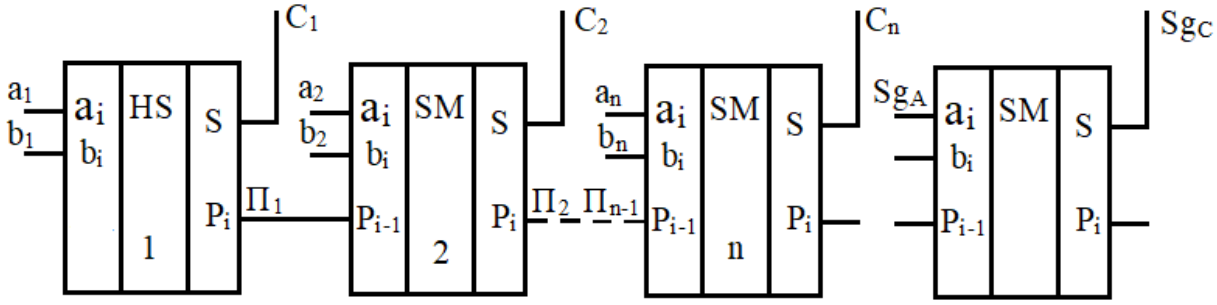


Рис. 5.2.

На ДСПК можно складывать только числа, имеющие одинаковые знаки, т.е. такой сумматор не может выполнять операцию алгебраического сложения.

Пусть заданы операнды:

$$[A]_{пр}=Sg_A a_1 a_2 \ldots a_n \,, \qquad [B]_{пр}=Sg_B b_1 b_2 \ldots b_n,$$

где Sg_A, Sg_B – соответственно содержимое знаковых разрядов изображений для A и B (символ происходит от английского слова sign – знак); a_i, b_i – цифровые разряды изображений. Если $Sg_A=Sg_B$, то сумма чисел будет иметь знак любого из слагаемых, а цифровая часть результата получится после сложения цифровых частей операндов.

Пример 5.1. Сложить числа A=0.1001 и B=0.0110 на ДСПК.

$[A]_{пр}$=0,1001	Sg_A=0	$[A]_{пр}$=1,1001	Sg_A=1
$[B]_{пр}$=0,0110	Sg_B=0	$[B]_{пр}$=1,0110	Sg_B=1
$[C]_{пр}$=0,1111	Sg_C=0	$[C]_{пр}$=1,1111	Sg_C=1

Если абсолютное значение суммы будет больше единицы, то имеет место переполнение разрядной сетки ДСПК. Как видно, признак переполнения разрядной сетки ДСПК – появление единицы переноса из старшего разряда цифровой части сумматора. В этом случае должен вырабатываться сигнал переполнения φ=1, по которому происходит корректировка масштабных коэффициентов или автоматическая остановка машины.

5.2.2 Двоичный сумматор дополнительного кода (ДСДК)

ДСДК – сумматор, оперирующий изображениями чисел в дополнительном коде. Характерная особенность ДСДК – наличие цепи поразрядного переноса из старшего разряда цифровой части в знаковый разряд. Условная схема ДСДК представлена на рис. 5.3.

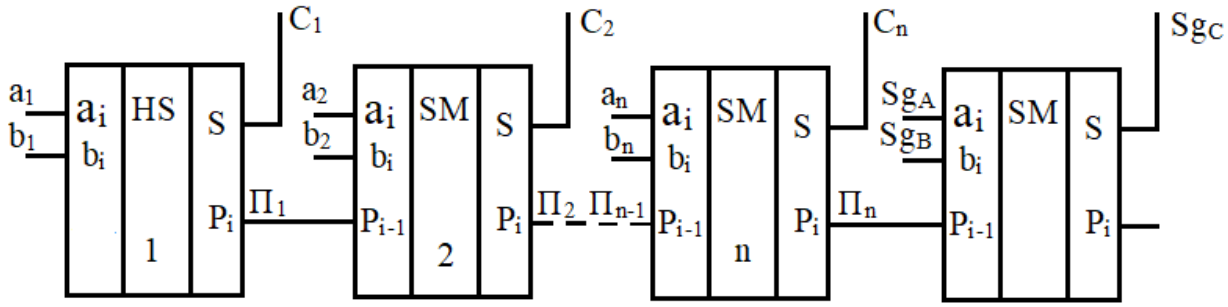


Рис.5.3

Теорема: ***Сумма дополнительных кодов чисел есть дополнительный код результата.***

Доказательство: Предполагая, что числа представлены в формате с фиксированной точкой, стоящей перед старшим разрядом, рассмотрим возможные случаи.

1) $A>0$, $B>0$, $A+B<1$.

Так как $[A]_Д=A$, $[B]_Д=B$, то $[A]_Д+[B]_Д=A+B=[A+B]_Д$ – положительный результат.

2) $A<0$, $B>0$, $|A|>B$.

Так как в этом случае $[A]_Д=A+q$, $[B]_Д=B$, то $[A]_Д+[B]_Д=A+B+q=[A+B]_Д$ – отрицательный результат (в знаковом разряде будет “1”).

3) $A<0$, $B>0$, $|A|<B$.

В этом случае имеем: $[A]_Д=A+q$, $[B]_Д=B$. И тогда $[A]_Д+[B]_Д=A+B+q$. Так как значение полученной в этом случае суммы больше q, то появляется единица переноса из знакового разряда, что равносильно изъятию из суммы q единиц. С учетом этого получаем результат: $[A]_Д+[B]_Д=A+B=[A+B]_Д$ – результат положительный, т.к. в знаковом разряде будет “0”.

4) $A<0$, $B<0$, $|A+B|<1$.

Здесь $[A]_Д=A+q$, $[B]_Д=B+q$. Тогда $[A]_Д+[B]_Д=A+B+q+q$. Появляется единица переноса из знакового разряда, что эквивалентно изъятию из суммы q единиц. В результате получается: $[A]_Д+[B]_Д=A+B+q=[A+B]_Д$ – отрицательный результат.

Таким образом, теорема справедлива для всех случаев, в которых не возникает переполнения разрядной сетки, что позволяет складывать машинные изображения чисел по правилам двоичной арифметики, не разделяя знаковую и цифровую части изображений.

Пример 5.2. Найти сумму чисел A=0.10110, B=0.00101, используя сумматор дополнительного кода

Решение. Складываются машинные изображения этих чисел

$$\begin{array}{rl} & [A]_д = 0.10110 \\ + & [B]_д = 0.00101 \\ \hline & [C]_д = 0.11011 \end{array}$$

Ответ: C=0.11011

Пример 5.3. Найти сумму A=-0.10110, B=0.00101, используя сумматор дополнительного кода

Решение

$$\begin{array}{rl} & [A]_д = 1.01010 \\ + & [B]_д = 0.00101 \\ \hline & [C]_д = 1.01111 \end{array}$$

Ответ: C=-0.10001.

Пример 5.4. Найти сумму чисел A=0.10110, B=-0.00101, используя сумматор дополнительного кода

Решение. Складывая машинные изображения чисел, представленных в дополнительном коде, получаем:

$$\begin{array}{rl} & [A]_д = 0.10110 \\ + & [B]_д = 1.11011 \\ \hline & [C]_д = 0.10001 \end{array}$$

Ответ: *C*=0.10001.

Пример 5.5. Найти сумму чисел A=-0.10110, B=-0.00101, используя сумматор дополнительного кода

Решение.

$$\begin{array}{rl} & [A]_д = 1.01010 \\ + & [B]_д = 1.11011 \\ \hline & [C]_д = 1.00101 \end{array}$$

Ответ: *C*=-0.11011.

5.2.3 Двоичный сумматор обратного кода (ДСОК)

ДСОК – сумматор, оперирующий изображением чисел в обратном коде. Характерная особенность ДСОК – наличие цепи кругового, или циклического, переноса из знакового разряда в младший разряд цифровой

части. Структурная схема ДСОК может быть представлена в виде, приведенном на рис. 5.4.

Теорема. ***Сумма обратных кодов есть обратный код результата.***

Доказательство. Аналогично предыдущему случаю рассмотрим основные случаи:

1) $A>0,\ B>0,\ A+B<1$.

Так как в этом случае $[A]_{об}=A,\ [B]_{об}=B$, то $[A]_{об}+[B]_{об}=A+B=[A+B]_{об}$ – результат положительный.

2) $A<0,\ B>0,\ |A|>B$.

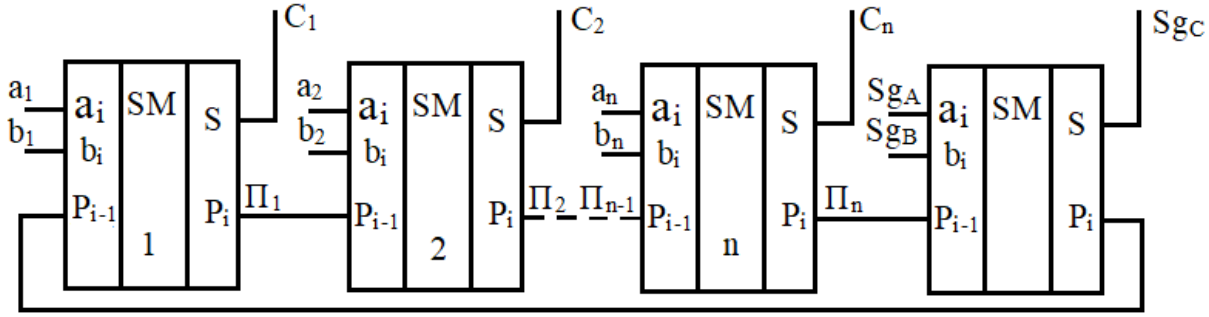


Рис. 5.4.

В этом случае имеем: $[A]_{об}=q-q^{-n}+A,\ [B]_{об}=B$. И тогда

$[A]_{об}+[B]_{об}=q+q^{-n}+A+B=[A+B]_{об}$ – результат отрицательный.

3) $A<0,\ B>0,\ |A|<B$.

В этом случае также имеем: $[A]_{об}=q-q^{-n}+A,\ [B]_{об}=B$, и

$$[A]_{об}+[B]_{об}=q-q^{-n}+A+B.$$

Но в данном случае сумма $[A]_{об}+[B]_{об}$ положительна, и правая часть выражения становится больше q, что вызывает появление единицы переноса из знакового разряда. Поскольку в ДСОК существует цепь переноса из знакового разряда в младший разряд (величина переноса из знакового разряда равна $q-q^{-n}$), то $[A]_{об}+[B]_{об}=[A+B]_{об}$ – результат положительный.

4) $A<0,\ B<0,\ |A+B|<1$.

Здесь имеем: $[A]_{об}=q-q^{-n}+A,\ [B]_{об}=q-q^{-n}+B$. Следовательно, сумма в этом случае будет иметь вид:

$$[A]_{об}+[B]_{об}=q-q^{-n}+A+q-q^{-n}+B=q-q^{-n}+q-q^{-n}+A+B.$$

Вполне очевидно, что здесь появляется единица переноса из знакового разряда, что эквивалентно изъятию из суммы величины $q-q^{-n}$. Тогда с учетом этого имеет место соотношение: $[A]_{об}+[B]_{об}=q-q^{-n}+A+B$ - результат отрицательный, т.е. $[A]_{об}+[B]_{об}=[A+B]_{об}$.

5) $|A|=B,\ A<0,\ B>0$.

Тогда в данном случае получаем: $[A]_{об}=q-q^{-n}+A,\ [B]_{об}=B$, и

$[A]_{об}+[B]_{об} = q-q^{-n} + A + B = q-q^{-n}$ – одно из изображений нуля в обратном коде.

Таким образом доказано, что на ДСОК машинные изображения чисел складываются по правилам, приведенным в таблице 5.2

Пример 5.6. Найти сумму двоичных чисел A=0.10110 и B=0.00101 используя сумматор обратного кода.

Решение. Складывая машинные изображения чисел, представленных в обратном коде, получаем:

$$\begin{array}{rl} & [A]_{об}=0.10110 \\ + & [B]_{об}=0.00101 \\ \hline & [C]_{об}=0.11011 \end{array}$$

Ответ: C=0.11011.

Пример 5.7. Найти сумму двоичных чисел A=-0.10110 и B=0.00101 используя сумматор обратного кода.

Решение.

$$\begin{array}{rl} & [A]_{об}=1.01001 \\ + & [B]_{об}=0.00101 \\ \hline & [C]_{об}=1.01110 \end{array}$$

Ответ: C=-0.10001.

Пример 5.8. Найти сумму чисел A=0.10110 и B=-0.00101 используя сумматор обратного кода.

Решение.

$$\begin{array}{rl} & [A]_{об}=0.10110 \\ + & [B]_{об}=1.11010 \\ \hline & [C]_{об}=0.10000 \\ & + \qquad\qquad 1 \quad \text{- единица переноса} \\ \hline & [C]_{об}=0.10001 \end{array}$$

Ответ: C=0.10001.

Пример 5.9. Найти сумму двоичных чисел A=-0.10110 и B=-0.00101 используя сумматор обратного кода.

Решение.

$$\begin{array}{rl} & [A]_{об}=1.01001 \\ + & \end{array}$$

$$\begin{array}{l} [B]_{об}=1.11010 \\ \hline [C]_{об}=1.00011 \\ + \qquad\qquad 1 \\ \hline [C]_{об}=1.00100 \end{array}$$ - единица переноса

Ответ: C=-0.11011.

5.3 Переполнение разрядной сетки

При сложении чисел одинакового знака, представленных в форме с фиксированной точкой, может возникнуть переполнение разрядной сетки. Каждый сумматор обладает своим признаком переполнения разрядной сетки, анализируя который можно корректировать дальнейшие действия.

1. Признаком переполнения разрядной сетки ДСПК является появление единицы переноса из старшего разряда цифровой части числа.

Пример 5.10. Выполнить сложение на ДСПК:

1. A=0.10110, B=0.10001; 2. A=-0.10110, B=-0.10001.

$$\begin{array}{l} + \quad [A]_{пр}=0.10110 \\ \quad\; [B]_{пр}=0.10001 \\ \hline [C]_{пр}\neq 0.00111 \end{array} \qquad \begin{array}{l} + \quad [A]_{пр}=1.10110 \\ \quad\; [B]_{пр}=1.10001 \\ \hline [C]_{пр}\neq 1.00111 \end{array}$$

1- единица переноса 1- единица переноса

2. Признаком переполнения разрядной сетки ДСДК и разрядной сетки ДСОК является получение отрицательного результата при сложении положительных чисел, и получение положительного результата при сложении отрицательных чисел, т.е. при переполнении РС знак результата противоположен знакам операндов.

Пример 5.11 . Выполнить сложение двоичных чисел на ДСДК:

1. A=0.10110, B=0.10001; 2. A=-0.10110, B=-0.10001.

$$\begin{array}{l} + \quad [A]_{д}=0.10110 \\ \quad\; [B]_{д}=0.10001 \\ \hline [C]_{д}\neq 1.00111 \end{array} \qquad \begin{array}{l} + \quad [A]_{д}=1.10110 \\ \quad\; [B]_{д}=1.10001 \\ \hline [C]_{д}\neq 0.11001 \end{array}$$

Пример 5.12. Выполнить сложение двоичных чисел на ДСОК:

1. A=0.10110, B=0.10001; 2. A=-0.10110, B=-0.10001.

$$\begin{array}{l} + \quad [A]_{об}=0.10110 \\ \quad\; [B]_{об}=0.10001 \\ \hline [C]_{об}\neq 1.00111 \end{array} \qquad \begin{array}{l} + \quad [A]_{об}=1.10110 \\ \quad\; [B]_{об}=1.10001 \\ \hline [C]_{об}\neq 0.10111 \end{array}$$

Для обнаружения переполнения разрядной сетки в составе цифрового автомата должны быть предусмотрены аппаратные средства, автоматически вырабатывающие признак переполнения – сигнал φ.

Чтобы обнаружить переполнение разрядной сетки ДСДК и ДСОК, вводится вспомогательный разряд в знаковую часть изображения числа, который называется **разрядом переполнения**. Такой формат представления данных называется **модифицированным**. На рис.5.5 изображен модифицированный формат представления числа.

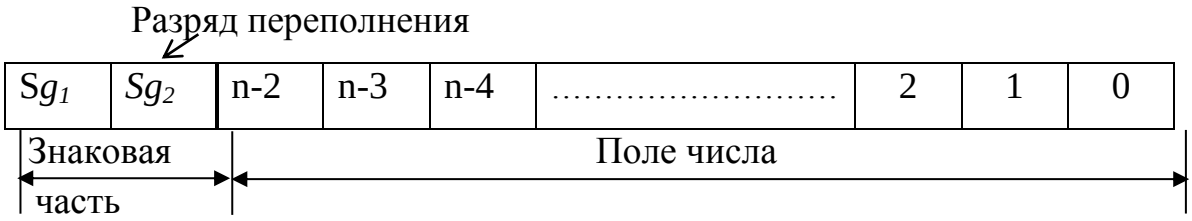


Рис. 5.5 Модифицированный формат представления данных

При представлении положительных чисел в модифицированном формате в знаковой части разрядной сетки помещаются нули (рис. 5.6 а), а при представлении отрицательных чисел – единицы (рис.5.6 б).

0	**0**	1	1	0	1	0	0	1	0	1	1

Рис. 5.6 а Представление положительного числа в модифицированном формате

1	**1**	1	1	0	1	0	0	1	0	1	1

Рис.5.6 б Представление отрицательного числа в модифицированном формате

Тогда в случае переполнения разрядной сетки при сложении чисел в знаковой части разрядной сетки будут находиться разные символы - | **0** | **1** | , или | **1** | **0** |. В этом случае функция φ=1, сигнализирующая о наличии переполнения разрядной сетки, будет равна единице, т.е.:

$$\varphi = \overline{Sg_1} \cdot Sg_2 \vee Sg_1 \cdot \overline{Sg_2} = (Sg_1 \oplus Sg_2) \qquad (5.1)$$

Пример 5.13. Выполнить сложение двоичных чисел на ДСОК, ДСПК, ДСДК, если:

1. A=0.10110, B=0.10001.

Решение

$$[A]^{\text{м}}_{\text{об}} = 00.10110 \rightarrow \text{модифицированное представление операнда } A$$
$$+\,[B]^{\text{м}}_{\text{об}} = 00.10001 \rightarrow \text{модифицированное представление операнда } B$$
$$[C]^{\text{м}}_{\text{об}} = 01.00111$$
$$\rightarrow 01 \text{ в знаковых разрядах} - \text{признак переполнения,}$$
т.е. φ=1.

2. A=-0.10110, B=-0.10001.

Решение

$[A]^{м}_{об} = 11.01001$ → модифицированное представление операнда A

$+[B]^{м}_{об} = 11.01110$ → модифицированное представление операнда B

$[C]^{м}_{об} = 10.11000$

→ 10 в знаковых разрядах – признак переполнения,

т.е. φ=1.

3. A=0.10110, B=0.10001.

Решение

$[A]^{м}_{д} = 00.10110$ → модифицированное представление операнда A

$+[B]^{м}_{д} = 00.10001$ → модифицированное представление операнда B

$[C]^{м}_{д} = 01.00111$

→ 01 в знаковых разрядах – признак переполнения,

т.е. φ=1.

4. A=-0.10110, B=-0.10001.

Решение

$[A]^{м}_{д} = 11.01010$ → модифицированное представление операнда A

$+[B]^{м}_{д} = 11.01111$ → модифицированное представление операнда B

$[C]^{м}_{д} = 10.11001$

→ 10 в знаковых разрядах – признак переполнения,

т.е. φ=1.

5.4 Сложение чисел в формате с плавающей точкой

Как было уже сказано, числа, представленные в формате с плавающей точкой, изображаются двумя частями – мантиссой и порядком. При операции алгебраического сложения действия, выполняемые над мантиссами и порядками, различны. Следовательно, в цифровом устройстве должны быть два раздельных устройства для обработки мантисс и для обработки порядка.

Для чисел с плавающей точкой справедливо условие (4.11):

$$(q^{-1} \le |m_A| < 1),$$

и всякий результат, не удовлетворяющий этому условию, должен быть приведен в соответствие с формулой (4.11). Операцию приведения результата (числа) к нормализованному виду называют операцией нормализации. Вполне очевидно, что операция нормализации числа состоит из проверки условия (4.11) и сдвига изображения мантиссы вправо или влево в пределах разрядной сетки.

Различают сдвиги: ***логический, циклический, арифметический.***

Логический сдвиг – смещение всей числовой последовательности слова, включая знаковый разряд. При этом в освободившихся *k* разрядов записываются нули.

Пример 5.14 . Имеем числовую последовательность:

1	1	0	1	1	0	0	1	1	1

После логического сдвига на 3 разряда вправо получаем:

0	0	0	1	1	0	1	1	0	0

Циклический сдвиг – смещение всей числовой последовательности, при которой значения разрядов, выходящих за пределы разрядной сетки снова вводятся в освобождающиеся позиции слова.

Пример 5.15. Имеем числовую последовательность:

1	1	0	1	1	0	0	1	1	1

После циклического сдвига на 3 разряда вправо получаем:

1	1	1	1	1	0	1	1	0	0

Арифметический сдвиг – сдвиг всей числовой последовательности без изменения позиции знака числа. Различают **простой** и модифицированный **арифметический** сдвиги.

Простой арифметический сдвиг эквивалентен умножению числа, представленного в определенной системе счисления, на основание этой системы счисления, возведенное в степень, равную величине сдвига.

Простой арифметический сдвиг влево для разных кодов осуществляется по-разному.

Для прямого кода влево сдвигается только цифровая часть, младшие разряды заполняются нулями. Например, для исходного числа, представленного в прямом коде

1	0	0	0	1	0	1	1	0	1	1

После сдвига на 3 разряда влево получим:

1	1	0	1	1	0	1	1	0	0	0

Для дополнительного кода (ДК) и обратного кода (ОК) сдвигается влево вся числовая последовательность, освобождающиеся младшие разряды для ДК заполняются нулями, для ОК – единицами. Для ДК выдвигающаяся знаковая единица теряется. Например, дополнительный код числа имеет вид:

1	0	0	1	1	1	0	0	1	1	1

После сдвига на 1 разряд влево получаем:

0	0	1	1	1	0	0	1	1	1	0

Для обратного кода выдвигающаяся знаковая единица переносится в младший разряд цифровой части (как для циклического сдвига влево). Например, обратный код числа имеет вид:

1	0	0	1	1	1	0	0	1	1	1

После сдвига на один разряд влево в этом случае получаем:

0	0	1	1	1	0	0	1	1	1	1

Простой арифметический сдвиг вправо:

Для прямого кода вправо сдвигается только цифровая часть.

Например, для числа, представленного в прямом коде

1	0	0	1	1	1	0	0	1	1	1

после сдвига на один разряд вправо получаем:

1	0	0	0	1	1	1	0	0	1	1

Для ДК и ОК вправо сдвигается вся числовая последовательность. Цифра знакового разряда перемещается в старший цифровой разряд и в то же время восстанавливается в знаковом разряде. Например, в исходном состоянии имеем:

1	0	0	1	1	1	0	0	1	1	1

После сдвига на один разряд вправо получаем:

1	1	0	0	1	1	1	0	0	1	1

Модифицированный арифметический сдвиг используется для чисел, представленных в формате с плавающей точкой. Для такого сдвига величина исходного числа не меняется, т.е. в этом случае наряду со смещением мантиссы числа одновременно изменяется и его порядок так, чтобы величина числа осталась без изменений.

5.5 Операция нормализации

Невыполнение условия (4.11) ($q^{-1} \le |m_A| < 1$) – есть нарушение нормализации числа. Приведение числа к нормализованной форме представления – операция нормализации, которая заключается в последовательном выполнении модифицированного арифметического сдвига, для которого каждый сдвиг мантиссы числа на один разряд влево будет сопровождаться вычитанием единицы из порядка числа. Такой

процесс продолжается до тех пор, пока число не примет нормализованный вид.

Так как условие (4.11) содержит два неравенства, то может быть нарушение как правого условия, так и левого (нарушение справа и слева).

Признак нарушения нормализации числа справа γ (т.е. $|m_A|\geq 1$) – наличие разных символов в знаковых разрядах сумматора.

$$\gamma = 1, \text{если } \overline{Sg_1}\&Sg_2 \vee Sg_1\&\overline{Sg_2} = 1 \qquad (5.2)$$

Соотношение γ=1 указывает на необходимость сдвига числа вправо на один разряд.

Признак нарушения нормализации числа слева δ (когда $|m_A|<q^{-1}$) – наличие одинаковых символов в разряде переполнения и в старшем разряде цифровой части сумматора (p_1).

$$\delta = 1, \text{если } Sg_2\&p_1 \vee \overline{Sg_2}\&\bar{p}_1 = 1. \qquad (5.3)$$

Соотношение δ=1 указывает на необходимость сдвига числа влево на один разряд.

Таким образом, операция нормализации числа состоит из совокупности сдвигов и проверки наличия признаков нарушения.

Рассмотрим сложение чисел, представленных в нормализованной форме: $C=A+B$, где $A=m_A p_A$, $B=m_B \cdot p_B$., где мантиссы m_A и m_B удовлетворяют условию (4.11).

1.Пусть $p_A=p_B$. Сложение мантисс осуществляется на соответствующем сумматоре по правилам, определяемым типом сумматора. Если после сложения, мантисса результата удовлетворяет условию нормализации, то к этому результату приписывается порядок любого из операндов. В противном случае необходимо выполнить нормализацию числа.

Пример 5.16. На ДСДК, включающим 6 разрядов мантиссы и 4 разряда для порядка, выполнить действие: $C=A+B$, если $A=0.1001\cdot 2^{-4}$ и $B=-0.1101\cdot 2^{-4}$.

Решение: $[m_A]^{\text{м}}_{\text{д}} = 00.1001; [p_A]_{\text{д}} = 1.100;$

$[m_B]^{\text{м}}_{\text{д}} = 11.0011; [p_B]_{\text{д}} = 1.100.$

Складывая мантиссы, получаем:

$$\begin{array}{r} [m_A]^{\text{м}}_{\text{д}} = 00.1001 \\ +[m_B]^{\text{м}}_{\text{д}} = 11.0011 \\ \hline [m_C]^{\text{м}}_{\text{д}} = 11.1100 \end{array}$$

Здесь получаем, что

$$\delta = Sg_2 \& p_1 \vee \overline{Sg}_2 \& \bar{p}_1 = 1 \cdot 1 \vee 1 \cdot 0 = 1 \vee 0 = 1.$$

$$\gamma = \overline{Sg}_1 \& Sg_2 \vee Sg_1 \& \overline{Sg}_2 = 0 \cdot 1 \vee 1 \cdot 0 = 0 \vee 0 = 0.$$

Т.е. имеется нарушение нормализации слева и, следовательно, необходимо осуществить модифицированный сдвиг влево. После сдвига на один разряд получаем:

$[m_C^*]_{\text{д}}^{\text{м}}$=11.1000. Т.е. здесь опять имеем δ=1

Одновременно со сдвигом осуществляется коррекция порядка, т.е. уменьшение порядка в данном случае на единицу ($[p_C^*]_{\text{д}} = [p_C]_{\text{д}} - 1$), т.е.

$$\begin{array}{r} [p_C]_{\text{д}} = 1.100 \\ +\underline{[-1]_{\text{д}} = 1.111} \\ [p_C^*]_{\text{д}} = 1.011 \end{array}$$

Т.к. после первого сдвига снова имеет место нарушение нормализации, то снова нужна коррекция результата, т.е. еще раз осуществляется сдвиг мантиссы и коррекция порядка:

$[m_C^{**}]_{\text{д}}^{\text{м}}$=11.0000. (Теперь получаем, что γ=0, δ=0).

$$\begin{array}{r} [p_C^*]_{\text{д}} = 1.011 \\ +\underline{[-1]_{\text{д}} = 1.111} \\ [p_C^{**}]_{\text{д}} = 1.010 \end{array}$$

Теперь можно записать результат: $[m_C]_{\text{д}}^{\text{м}}=[m_C^*]_{\text{д}}^{\text{м}} = 11.0000$; $[p_C^{**}]_{\text{д}} = 1.010$. C=-1.0000·10^{-110}=-2$^{-6}{}_{10}$.

Пример 5.17. На ДСОК, включающим 6 разрядов мантиссы и 4 разряда для порядка, выполнить действие: C=A+B, если A=0.1001·2^{-4} и B=-0.1101·2^{-4}. (т.е. выполним сложение чисел из предыдущего примера на ДСОК).

Решение: $[m_A]_{\text{об}}^{\text{м}} = 00.1001$; $[p_A]_{\text{об}} = 1.011$;

$[m_B]_{\text{об}}^{\text{м}} = 11.0010$; $[p_B]_{\text{об}} = 1.011$.

Складывая мантиссы, получаем:

$$\begin{array}{r} [m_A]_{\text{д}}^{\text{м}} = 00.1001 \\ +\underline{[m_B]_{\text{д}}^{\text{м}} = 11.0010} \\ [m_C]_{\text{д}}^{\text{м}} = 11.1011 \end{array}$$

Здесь получаем, что

$$\delta = Sg_2 \& p_1 \vee \overline{Sg}_2 \& \bar{p}_1 = 1 \cdot 1 \vee 0 \cdot 0 = 1 \vee 0 = 1.$$

$$\gamma = \overline{Sg}_1 \& Sg_2 \vee Sg_1 \& \overline{Sg}_2 = 0 \cdot 1 \vee 1 \cdot 0 = 0 \vee 0 = 0.$$

Следовательно, необходим сдвиг влево. После сдвига получаем: $[m_C^*]_{\text{д}}^{\text{м}}$=11.0111, и

$$\begin{array}{r} [p_C]_{\text{об}} = 1.011 \\ +\underline{[-1]_{\text{об}} = 1.110} \\ [p_C^*]_{\text{об}} = 1.010 \end{array}$$

Теперь нарушения нормализации числа нет, и конечный результат будет иметь вид:

$[m_C]_{\text{об}}^{\text{м}}=[m_C^*]_{\text{об}}^{\text{м}} = 11.0111;\ \ [p_C^*]_{\text{об}} = 1.010$. C=-0.1·10^{-101}=-2^{-6}_{10}.

2. Пусть теперь $p_A \neq p_B$. Для операции сложения чисел необходимым условием является соответствие разрядов операндов друг другу. Следовательно, нужно уровнять порядки, что повлечет за собой, естественно, временное нарушение нормализации одного из операндов. Выравнивание порядков означает, что порядок меньшего числа надо увеличить на величину $\Delta p=|p_A-p_B|$, что в свою очередь означает сдвиг мантиссы меньшего числа вправо на количество разрядов, равное Δp. Если $p_A-p_B>0$, то $p_A>p_B$ и сдвигается на Δp разрядов вправо мантисса числа B; если же $p_A-p_B<0$, то $p_A<p_B$ и сдвигается вправо на Δp разрядов мантисса числа A. Далее выполняется операция сложения чисел по вышеописанным правилам.

Операция сложения и вычитания чисел в форме с плавающей запятой осуществляется во всех современных машинах по изложенным выше правилам.

Пример 5.18. На 10-разрядном ДСОК, включающим 6 разрядов мантиссы и 4 разряда для порядка, выполнить действие:

$C=A+B$, если A=0.1001·2^{-2} и B=-0.1011·2^{-3}.

Решение:

$[m_A]_{\text{об}}^{\text{м}} = 00.1001;\ \ [p_A]_{\text{об}} = 1.101;$

$[m_B]_{\text{об}}^{\text{м}} = 11.0100;\ \ [p_B]_{\text{об}} = 1.100.$

$[\Delta p]_{\text{об}} = [p_A]_{\text{об}} - [p_B]_{\text{об}} = [p_A]_{\text{об}}(-[p_B]_{\text{об}}) = 1.101 + 0.011 = 0.001.$

Т.к. $\Delta p > 0$, следовательно, $p_A>p_B$, и нужно сдвинуть мантиссу числа B на один разряд. После сдвига получаем: $[m_B]_{\text{об}}^{\text{м}} = 11.1010$. Складывая мантиссы, получаем:

Складывая мантиссы, получаем:

$$\begin{array}{r} [m_A]_{\text{об}}^{\text{м}} = 00.1001 \\ +\underline{[m_B]_{\text{об}}^{\text{м}} = 11.1010} \\ [m_C]_{\text{об}}^{\text{м}} = 00.0011 \end{array}$$

$$\begin{array}{l} +\ 1 - \text{единица переноса из знакового разряда} \\ \hline [m_C]^{\text{м}}_{\text{об}} = 00.0100 \end{array}$$

Здесь необходимо выполнить операцию нормализации результат, т.к. имеет место нарушение нормализации слева:

$$\delta = Sg_2 \& p_1 \vee \overline{Sg}_2 \& \bar{p}_1 = 0 \cdot 0 \vee 1 \cdot 1 = 0 \vee 1 = 1.$$

$$\gamma = \overline{Sg}_1 \& Sg_2 \vee Sg_1 \& \overline{Sg}_2 = 1 \cdot 0 \vee 0 \cdot 1 = 0 \vee 0 = 0.$$

Следовательно, необходим сдвиг влево. После сдвига получаем: $[m_C^*]^{\text{м}}_{\text{об}}$=00.1000, и

$$\begin{array}{l} [p_C]_{\text{об}} = 1.101 \\ +\ [-1]_{\text{об}} = 1.110 \\ \hline [p_C^*]_{\text{об}} = 1.100 \end{array}$$

Теперь δ=0, γ=0, и можно записать результат:

$[m_C]^{\text{м}}_{\text{об}} = [m_C^*]^{\text{м}}_{\text{об}}$=00.1000; $[p_C]_{\text{об}} = [p_C^*]_{\text{об}}$=1.100;

m_C =0.1000; p_C=-011; C=$0.1 \cdot 10^{-11}{}_2$.

5.6 Оценка точности выполнения арифметических операций

Выбор системы счисления и длины разрядной сетки, а также формы представления числа в машине тесно связаны с обеспечением заданной точности вычислений. Важное значение имеет оценка точности арифметических вычислений в форме с фиксированной и плавающей точкой. При операциях сложения и вычитания чисел, представленных в формате с фиксированной точкой, можно считать, что они выполняются точно.

Для чисел, представленных в форме с плавающей точкой, при операциях сложения и вычитания необходимо выравнивать порядки, что ведет к потере значений некоторых разрядов мантиссы при сдвиге. Поэтому, при нормализованной форме представления чисел сама операция алгебраического сложения чисел также является источником погрешностей.

Таким образом, причинами погрешностей вычислений в ЭВМ являются:

- неточное задание исходных данных;
- погрешности перевода (представления) чисел;
- погрешности, вызванные ограничением ДРС;
- использование приближенных методов вычислений;
- округление результатов элементарных операций;
- сбои в работе ЭВМ.

5.6.1 Погрешности выполнения арифметических операций

Произведем действия над числами A и B, заданными с абсолютными погрешностями ΔA и ΔB соответственно: $A=[A]+\Delta A$, $B=[B]+\Delta B$.

При выполнении операции сложения имеем:

$$A+B=[A]+\Delta A+[B]+\Delta B=[A]+[B]+(\Delta A+\Delta B)=[A]+[B]+\Delta(A+B), \quad (5.4)$$

где $\Delta(A+B)=\Delta A+\Delta B$ – абсолютная погрешность суммы.

При выполнении операции вычитания:

$$A-B=[A]+\Delta A-[B]-\Delta B=[A]-[B]+(\Delta A-\Delta B)=[A]-[B]+\Delta(A-B), \quad (5.5)$$

где $\Delta(A-B)=\Delta A-\Delta B$ – абсолютная погрешность разности.

При выполнении операции умножения;

$$A\cdot B=([A]+\Delta A)\cdot([B]+\Delta B)=[A]\cdot[B]+B\cdot\Delta A+[A]\cdot\Delta B+\Delta A\cdot\Delta B. \quad (5.6)$$

Так как $\Delta A\cdot\Delta B$ на два порядка меньше чисел A и B, то этим произведением можно пренебречь, и, следовательно,

$$A\cdot B\approx[A]\cdot[B]+B\cdot\Delta A+[A]\cdot\Delta B=[A]\cdot[B]+\Delta(A\cdot B), \quad (5.7)$$

где $\Delta(A\cdot B)$- абсолютная погрешность произведения.

При выполнении операции деления:

$$\frac{A}{B}=\frac{[A]+\Delta A}{[B]+\Delta B}=\frac{[A]+\Delta A}{[B]}\cdot\left(\frac{1}{1+{}^{\Delta B}/_{B}}\right). \quad (5.8)$$

Второй сомножитель разложим в ряд и после преобразований получим:

$$\frac{A}{B}=\frac{[A]}{[B]}-\frac{[A]\cdot\Delta B}{[B]^2}+\frac{[A]\cdot\Delta B^2}{[B]^3}+\frac{\Delta A}{[B]}-\frac{\Delta A\cdot\Delta B}{[B]^2}+\cdots \quad (5.9)$$

Пренебрегая членами второго порядка малости, получаем:

$$\frac{A}{B}=\frac{[A]}{[B]}+\frac{\Delta A}{[B]}-\frac{\Delta A\cdot\Delta B}{[B]^2}=\frac{[A]}{[B]}+\Delta\left(\frac{A}{B}\right), \quad (5.10)$$

где $\Delta\left(\frac{A}{B}\right)$ - абсолютная погрешность частного.

Аналогично можно вывести выражения для относительных погрешностей:

$$\delta_{\pm}=\frac{[A]}{[A]+[B]}\cdot\frac{\Delta A}{[A]}\pm\frac{[B]}{[A]+[B]}\cdot\frac{\Delta B}{[B]} \quad (5.11)$$

$$\delta_{A\cdot B}=\frac{\Delta A}{[A]}+\frac{\Delta B}{[B]}; \quad (5.12)$$

$$\delta_{\frac{A}{B}}=\frac{\Delta A}{[A]}-\frac{\Delta B}{[B]} \quad (5.13)$$

5.6.2 Погрешности округления

Если предположить, что исходная информация не содержит никаких ошибок и все вычислительные процессы конечны и не приводят к

ошибкам, то все равно присутствует третий тип ошибок – ошибки округления.

Пример 5.19 . Имеем гипотетическую машину с пятиразрядной сеткой. Необходимо сложить 9.2654 и 7.1625. Примем, что эти числа точные. Но их сумма, равная 16.4279, содержит шесть разрядов. Поэтому шестиразрядный результат будет округлен до значения 16.428.

Погрешность округления имеет смысл только для действительных чисел, т.к. ЭВМ автоматически выравнивает порядки действительных чисел при сложении и вычитании.

Для чисел, представленных в форме с плавающей точкой, справедливы выражения: $A_q = m_A \cdot q^k$ и $q^{-1} \le |m_A| < 1$.

Если для представления мантиссы используется только *n* разрядов, то изображение числа разбивается на две части:

$$A_q = [m_A] \cdot q^n + [A_0] \cdot q^{r-n}, \qquad (5.14)$$

где $[A_0] \cdot q^{r-n} = A_0$ - ‘хвост’ числа, не попавший в разрядную сетку.

В зависимости от того, как учитывается величина A_0 в машинном изображении, существует несколько способов округления.

1. ***Отбрасывание A_0.*** При этом возникает относительная погрешность:

$$\delta_{\text{окр}} = \frac{|A_0| \cdot q^{k-n}}{|m_A| \cdot q^k}. \qquad (5.15)$$

Т.к. $q^{-1} \le |m_A| < 1$ и $0 \le |A_0| < 1$, то

$$\delta_{\text{окр}} = \frac{1 \cdot q^{k-n}}{q^{-1} \cdot q^k} = q^{-(n-1)}, \qquad (5.16)$$

т.е. не зависит от величины самого числа, а зависит только от количества разрядов в машине для любой системы счисления.

2. ***Симметричное округление***. При этом производится анализ величины A_0. Принимается, что

$$[A] = \begin{cases} [m_A] \cdot q^n, & \text{если } |A_0| < q^{-1}, \\ [m_A] \cdot q^n + q^{k-n}, & \text{если } |A_0| \ge q^{-1}. \end{cases} \qquad (5.17)$$

При условии $|A_0| \ge q^{-1}$ производится прибавление единицы к младшему разряду мантиссы. Абсолютная погрешность округления при этом будет равна:

$$\Delta_{\text{окр}} = \begin{cases} |A_0| \cdot q^{k-n}, \\ |1 - A_0| \cdot q^{k-n}. \end{cases} \qquad (5.18)$$

Максимально возможное значение модуля абсолютной погрешности равно $0.5q^{k-n}$. Способ симметричного округления наиболее часто используется на практике.

3. ***Округление по дополнению***. В этом случае для округления берется информация, содержащая в (*n+1*)-м разряде. В случае двоичной системы счисления, если в (*n+1*)-м разряде содержится единица, в n разряд добавляется единица; если в (*n+1*)-м разряде содержится нуль, содержимое разрядов правее n-го отбрасывается.

4. ***Случайное округление***. Для такого округления необходимо иметь датчик случайных величин (1 или 0), который выдает единицу в самый младший разряд машинного изображения числа.

Лекция 6.
Кодирование информации

Кодирование информации – это представление сведений в той или иной стандартной форме.

Кодирование – это процесс представления информации в виде кода – набора условных обозначений для представления информации.

Кодирование – это переход от одной формы представления информации к другой, более удобной для хранения, передачи или обработки. Кодирование имеет главной целью сохранение информации.

Необходимым компонентом любого информационного процесса является кодирование и декодирование информации. Кодирование и декодирование должны быть взаимно-обратными операциями.

В каждой ЭВМ для представления информации используется набор символов. Каждый символ этого набора представляется определенным числом (кодом). Причем в разных типах ЭВМ используются разные способы кодирования. (т.е. одним и тем же символам сопоставляются разные коды).

Символьная (алфавитно-цифровая) информация хранится и обрабатывается в ЭВМ в форме цифрового кода, т. е. каждому символу ставится в соответствие отдельное бинарное слово-код. Необходимый набор символов, предусмотренный в конкретной ЭВМ, обычно включает в себя:

- - буквенно-цифровые знаки алфавита (алфавитов);
- - специальные знаки (пробел, скобки, знаки препинания и др.);
- - знаки операций.

Кроме того, в состав набора входят управляющие символы, соответствующие определенным функциям.

Так как ЭВМ может обрабатывать информацию, представленную только в числовой форме, то любая информация (текстовая, графическая, звуковая и др.) преобразуется в числовую информацию. С этой целью осуществляется кодирование входной информации для последующей ее обработки в ЭВМ. Кроме этого, кодирование входной информации имеет и другие цели, как, например:

- защита от несанкционированного доступа;
- уменьшение вероятности искажения при передаче из-за воздействия случайных помех;

- удобство обработки (на ЭВМ - двоичный код символов, различные способы представления чисел).

6.1. Кодирование текстовой информации

Текстовая информация состоит из букв, цифр, специальных символов (+, & и т. д.). В некоторых языках используются иероглифы. Согласно одному из принципов фон Неймана, для хранения информации в памяти ЭВМ используется двоичный код.

В США первоначально кодировали 26 букв латиницы, цифры 0, 1, 2, ..., 9 и несколько десятков специальных символов, всего чуть больше сотни. Код из *n* бит позволяет закодировать 2^n символов. Так как 2^7=128, то был принят семиразрядный двоичный код, получивший название **ASCII** (***American standard code for information interchange*** - американский стандартный код для обмена информацией).

sp 32	! 33	« 34	# 35	$ 36	% 37	& 38	‘ 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	del 127

Рис. 6.1 Базовая таблица ASCII

Для того чтобы кодировать кроме латиницы ещё и национальные алфавиты, к ASCII добавили слева один разряд, расширив таблицу кодирования от 128 до 256 символов. Первые 128 символов (от 0 до 127) – это цифры, прописные и строчные буквы латинского алфавита, управляющие символы. Вторая половина кодовой таблицы (от 128 до 255) предназначена для национальных символов (в том числе кириллицы), математических символов и так называемых псевдографических символов, которые используются для рисования рамок.

Для представления букв русского алфавита в рамках ASCII первоначально был разработан **КОИ-7** (Код Обмена Информацией 7-битный). Расположение символов во второй половине таблицы этой

кодировки резко отличается от принятого фирмой IBM, что затрудняет использование зарубежного программного обеспечения на отечественных ЭВМ. В связи с этим он практически не применяется, а на отечественных ПК введена так называемая ГОСТ-альтернативная кодировка, главное достоинство которой - расположение символов псевдографики на тех же местах, что и в кодировке IBM.

Другая распространенная кодировка носит название **КОИ-8** - (двоичный восьмибитовый Код Обмена Информацией) является стандартной русской кодовой таблицей на компьютерах, работающих под управлением операционной системы UNIX. Сегодня кодировка КОИ-8 имеет широкое распространение в компьютерных сетях на территории России и в некоторых службах российского сектора Интернета. В частности, в России она де-факто является стандартной в сообщениях электронной почты и телеконференций. Но даже 8-битная кодировка недостаточна для кодирования всех символов, которые хотелось бы иметь в расширенном алфавите. Все препятствия могут быть сняты при переходе, например, на 16-битную кодировку **UNICODE**, допускающую в этом случае 65536 кодовых комбинаций - этого достаточно для размещения в одной таблице всех широко употребляемых языков.

─ 128	│ 129	┌ 130	┐ 131	└ 132	┘ 133	├ 134	┤ 135	┬ 136	┴ 137	┼ 138	▀ 139	▄ 140	▌ 141	▐ 142	▀ 143
▒ 144	▒ 145	▓ 146	147	■ 148	• 149	√ 150	≈ 151	≤ 152	≥ 153	nbsp 154	155	° 156	² 157	· 158	÷ 159
═ 160	║ 161	╒ 162	ё 163	╓ 164	╔ 165	╕ 166	╖ 167	╗ 168	╘ 169	╙ 170	╚ 171	╛ 172	╜ 173	╝ 174	╞ 175
╟ 176	╠ 177	╡ 178	Ё 179	╢ 180	╣ 181	╤ 182	╥ 183	╦ 184	╧ 185	╨ 186	╩ 187	╪ 188	╫ 189	╬ 190	© 191
ю 192	а 193	б 194	ц 195	д 196	е 197	ф 198	г 199	х 200	и 201	й 202	к 203	л 204	м 205	н 206	о 207
п 208	я 209	р 210	с 211	т 212	у 213	ж 214	в 215	ь 216	ы 217	з 218	ш 219	э 220	щ 221	ч 222	ъ 223
Ю 224	А 225	Б 226	Ц 227	Д 228	Е 229	Ф 230	Г 231	Х 232	И 233	Й 234	К 235	Л 236	М 237	Н 238	О 239
П 240	Я 241	Р 242	С 243	Т 244	У 245	Ж 246	В 247	Ь 248	Ы 249	З 250	Ш 251	Э 252	Щ 253	Ч 254	Ъ 255

Рис. 6.2 Расширенная таблица кодировки КОИ-8

Нужно помнить о трех особенностях алфавита в кодовой таблице и их следствия:

1. Прописные и строчные буквы представлены разными кодами, т.е. “A” и “a” – разные объекты;
2. При упорядочивании слов по алфавиту сравниваются между собой десятичные коды букв. Поэтому, чтобы избежать недоразумений, если не указано “нечувствителен к регистру”, используйте только латинский или русский алфавит и только прописные или только строчные первые буквы. Необходимо помнить, что любая цифра “меньше” любой буквы, код латинских букв “меньше” чем русских;
3. Многие латинские и русские буквы имеют визуально неразличимое начертание, но разные коды.

В настоящее время имеется большое количество восьмибитных кодировок, отличающихся только второй половиной таблицы кодов. Для кириллицы используется несколько кодировок. Например, кодирования русской буквы "А" в кодировках КОИ-8 и Windows-1251:

КОИ-8: "А" $\rightarrow$ $E1_{16} = 1110\ 0001_2 = 225_{10}$,

Windows-1251: "А" $\rightarrow$ $C0_{16} = 1100\ 0000_2 = 192_{10}$.

Любая цифра в обеих кодировках имеет один код. Например, код цифры 6:

"6" $\rightarrow$ $36_{16} = 0011\ 0110_2 = 54_{10}$.

Во многих практических случаях, например, при записи иероглифов, используется гораздо больше, чем 256 символов, поэтому были разработаны многобайтовые кодировки.

6.2 Кодировки UNICODE

UNICODE (***Юникод***) - стандарт кодирования символов, разработанный в 1991 году и предназначенный для кодирования большого, практически неограниченного (1 112 064) количества различных символов. Стандарт состоит из двух частей (разделов):

- **UCS** – универсальный набор символов (***universal character set***);
- **UTF** – семейство кодировок для кодирования текстов (***unicode transformation format***).

Универсальный набор символов UCS задаёт однозначное соответствие символов кодам - элементам кодового пространства, представляющим неотрицательные целые числа. Семейство кодировок определяет машинное представление последовательности кодов UCS.

Коды в стандарте Unicode разделены на несколько областей. Область с кодами от U+0000 до U+007F содержит символы набора ASCII с соответствующими кодами. Далее расположены области знаков различных письменностей, знаки пунктуации и технические символы. Под символы кириллицы выделены области знаков с кодами от U+0400 до U+052F, от U+2DE0 до U+2DFF, от U+A640 до U+A69F. Часть кодов зарезервирована для использования в будущем

Кодовое пространство

Хотя формы записи UTF-8 и UTF-32 позволяют кодировать до 2^{31} (2 147 483 648) кодовых позиций, было принято решение использовать лишь 1 112 064 для совместимости с UTF-16. Впрочем, даже и этого на текущий момент более чем достаточно - в версии 6.0 используется чуть менее 110 000 кодовых позиций (109 242) графических и 273 прочих символов.

Кодовое пространство UCS разделено на 17 плоскостей (табл. 6.1) по 2^{16} (65536) символов

Таблица 6.1. Плоскости Юникода

№ плоскости	Название	Диапазон символов
0	Базовая многоязыковая плоскость (Basic multilingual plane, BMP)	U+0000…U+FFFF
1	Дополнительная многоязыковая плоскость (Supplementary multilingual plane, SMP)	U+10000…U+1FFFF
2	Дополнительная иероглифическая плоскость (Supplementary ideographic plane, SIP)	U+20000…U+2FFFF
3	Третичная иероглифическая плоскость (Tertiary ideographic plane, TIP)	U+30000…U+3FFFF
4—13	не используются	U+40000…U+DFFFF
14	Дополнительная плоскость особого назначения (Supplementary special-purpose plane, SSP)	U+E0000…U+EFFFF
15—16	Дополнительные области для частного использования (Supplementary private use area,	U+F0000…U+10FFFF

	SPUA-A/B)	

Нулевая плоскость называется базовой и содержит символы наиболее употребительных письменностей. Остальные плоскости - дополнительные. Первая плоскость используется в основном для исторических письменностей, вторая - для редко используемых иероглифов китайского письма (ККЯ), третья зарезервирована для архаичных китайских иероглифов. Плоскости 15 и 16 выделены для частного употребления.

Для обозначения символов ***UNICODE*** используется запись вида «U+xxxx» (для кодов 0000…FFFF), или «U+xxxxx» (для кодов 10000…FFFFF), или «U+xxxxxx» (для кодов 100000…10FFFF), где x — шестнадцатеричная цифра. Например, символ «д» (U+0436) имеет код $0434_{16} = 1076_{10}$, а символ «я» (U+044F) имеет код $044F_{16}=1103_{10}$.

Для кодирования русскоязычных текстов используется базовая плоскость с диапазоном кодов U+0000…U+FFFF.

Коды 0000 – 007F $(0000 – 0127)_{10}$ – это коды ASCII.

Коды 0400 – 04FF – кириллица.

Коды 0410 – 0427 – буквы А – Я, кроме Ё.

Коды 0430 – 044F – буквы а – я, кроме ё.

Код 0401 – буква Ё.

Код 0451 – буква ё.

Модифицирующие символы

Графические символы в Юникоде делятся на ***протяжённые*** и ***непротяжённые***. Непротяжённые символы при отображении не занимают дополнительного места в строке. К примеру, к ним относятся знак ударения. Протяжённые и непротяжённые символы имеют собственные коды, но последние не могут встречаться самостоятельно. Протяжённые символы называются базовыми (англ. base characters), а непротяженные - модифицирующими (англ. combining characters). Например символ «Й» (U+0419) может быть представлен в виде базового символа «И» (U+0418) и модифицирующего символа «̆» (U+0306).

Способы представления

Коды **UCS** из разных плоскостей имеют разную длину и не все кратны байту. Передача и хранение данных в современных ЭВМ и сетях производится побайтно. Память адресуется также побайтно. Поэтому для использования в ЭВМ коды UCS преобразуются в кратные байту коды в одну из форм кодировок UTF (англ. Unicode Transformation Format, UTF): UTF-8, UTF-16 (UTF-16BE, UTF-16LE) и UTF-32 (UTF-32BE, UTF-32LE).

Была разработана также форма представления UTF-7 для передачи по семибитным каналам, но из-за несовместимости с ASCII она не получила распространения и не включена в стандарт.

UTF-8

UTF-8 — представление Юникода, обеспечивающее наилучшую совместимость со старыми системами, использовавшими 8-битные символы. Текст, состоящий только из символов с номером меньше 128, при записи в UTF-8 превращается в обычный текст ASCII. И наоборот, в тексте UTF-8 любой байт со значением меньше 128 изображает символ ASCII с тем же кодом. Остальные символы Юникода изображаются последовательностями длиной от двух до шести байт (на деле, только до четырех байт, поскольку в Юникоде нет символов с кодом больше $10FFFF_{16}$, и вводить их в будущем не планируется), в которых первый байт всегда имеет вид 11xxxxxx, а остальные — 10xxxxxx.

Символы UTF-8 получаются из Unicode следующим образом:

Unicode	UTF-8	Представленные символы
0x00000000 - 0x0000007F	0xxxxxxx	ASCII, в том числе английский алфавит, простейшие знаки препинания и арабские цифры
0x00000080 - 0x000007FF	110xxxxx 10xxxxxx	кириллица, расширенная латиница, арабский алфавит, армянский алфавит, греческий алфавит, еврейский алфавит и коптский алфавит; сирийское письмо, тана, нко; Международный фонетический алфавит; некоторые знаки препинания
0x00000800 - 0x0000FFFF	1110xxxx 10xxxxxx 10xxxxxx	все другие современные формы письменности, в том числе грузинский алфавит, индийское, китайское, корейское и японское письмо; сложные знаки препинания; математические и другие специальные символы
0x00010000 - 0x001FFFFF	11110xxx 10xxxxxx 1 0xxxxxx 10xxxxxx	музыкальные символы, редкие китайские иероглифы, вымершие

		формы письменности
	111111xx	служебные символы c, d, e, f

Несмотря на то, что UTF-8 позволяет указать один и тот же символ несколькими способами, только наиболее короткий из них правильный. Остальные формы, называемые overlong sequence, отвергаются по соображениям безопасности.

Правила записи кода одного символа в UTF-8

Для понимания способа преобразования кода UCS в UTF-8 необходимо отметить, что длина кода символа в UCS подсчитывается в битах. Длина кода в диапазоне U+0000…U+007F равна семи битам, т. е. взят самый большой код из диапазона, отброшены левые нули, получившийся код $7F_{16}$ переведён в двоичную систему счисления: $7F_{16}$ = $111\ 1111_2$. Таким образом, в UCS коды ASCII, как и в изначальной кодировке ASCII, семибитные. Аналогично коды кириллицы в UCS имеют длину 11-бит:

U+04FF → 4FF → $100\ 1111\ 1111_2$.

Наиболее широкое применение получила кодировка UTF-8 (табл. 6.2).

Таблица 6.2. Соответствие кодов UCS и UTF-8

Диапазон кодов UCS	UTF-8	
	Длина (байт)	Диапазон кодов
0000-007F	1	0-7F
0080-07FF	2	C280 -DFBF
0800-FFFF	3	E0A080 – EFBFBF
10000-10FFFF	4	F0908080 - F48FBFBF

Итак:

1. Если размер символа в кодировке UTF-8 = 1 байт:

Код имеет вид (0aaa aaaa), где «0» - просто ноль, остальные биты «a» - это код символа в кодировке ASCII;

2. Если размер символа в кодировке в UTF-8 >1байт (т.е. от 2 до 6):

2.1 Первый байт содержит количество байт символа, закодированное в единичной системе счисления:

2 — 11

3 — 111

4 — 1111

5 — 1111 1

6 — 1111 11

2.2 «0» - бит терминатор, означающий завершение кода размера.

2.3 далее идут значащие байты кода, которые имеют вид (10xx xxxx), где «10» — биты признака продолжения, а «x» — значащие биты.

В общем случае коды в UTF-8 получаются из кодов UCS с помощью шаблонов (табл. 6.3). Обозначим через X двоичную цифру кода UCS, тогда любой *n*-битный код в общем виде будет обозначен как последовательность из *n* бит. Например, однобайтный код UCS будет записываться в виде XXXXXXX. Для перевода в UTF-8 формально к этому коду слева добавляется 0.

Таблица 6.3

(1 байт) 0xxx xxxx
(2 байта) 110x xxxx 10xx xxxx
(3 байта) 1110 xxxx 10xx xxxx 10xx xxxx
(4 байта) 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx
(5 байт) 1111 10xx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx
(6 байт) 1111 110x 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx

Длина кода в UTF-8, кодируемых n байтами UTF-8, определяется по формуле:

C=7 при n=1

C=n·5+1 при n>1

Количество байт UTF-8	Количество значащих бит
1	7
2	11
3	16
4	21
5	26
6	31

Пример 6.1. Перевод однобайтного UCS кода. Переведём, например, код числа 1. Код единицы в UCS U+0031, или в виде семи бит 011 0001. Формально для перевода в UTF-8 семибитный код добавляется до байта нулём слева:

$0011\ 0001_2 \rightarrow 31_{16}$.

Пример 6.2. Перевод двухбайтного UCS кода. Рассмотрим, как кодируется в UTF-8 буква Ж. Её код в UCS - 1046_{10} или 0416_{16} или 10000

010110_2 . Код UCS в двоичном виде разбивается на две части: пять левых бит 10000 и шесть правых 010110. Левая часть дополняется до байта признаком **110** двухбайтного кода UTF-8: **110**10000. В признаке 110 две единицы - это число 2 в единичной системе счисления. К правой части приписываются два бита **10** признака продолжения многобайтного кода: **10**010110. Окончательно код буквы Ж в UTF-8 выглядит так:

$11010000\ 10010110_2$ или $D0\ 96_{16}$

Пример 6.3. Перевод трёхбайтного UCS кода. Переведём в UTF-8 код U+266B нотного символа ♫. Код U+266B лежит в диапазоне

0800 < 266B < FFFF,

поэтому код в UTF-8 - трёхбайтный (см. табл.6.2). Переведём U+266B в двоичную систему счисления и разобьём двоичный код на три части

$266B_{16} \rightarrow 0010\ \ 011001\ \ 101011_2$

Дополнив первую часть до байта признаком **1110** трёхбайтного кода UTF-8, а вторую и третью части - признаком продолжения многобайтного кода, получим код нотного символа ♫ в кодировке UTF-8:

11100010 **10**011001 **10**101011$_2$→ E2 99 AB_{16}

Пример 6.4. Перевод четырёхбайтного UCS кода. Переведём в UTF-8 код U+1F31B символа 🌛 "Первая четверть луны с лицом". Код U+1F31B лежит в диапазоне

10000 < 1F31B < 10FFFF,

поэтому код в UTF-8 - четырёхбайтный (см. табл.4.2). Переведём U+1F31B в двоичную систему счисления и разобьём двоичный код на четыре части. Дополнив первую часть до байта признаком **11110** четырёхбайтного кода UTF-8, а вторую, третью и четвёртую части - признаком **10** продолжения многобайтного кода, получим код символа 🌛 в кодировке UTF-8:

11110000 **10**011111 **10**001100 **10**011011$_2$ → F0 9F 8C $9B_{16}$

Кодировку UTF-8 необходимо применять только к текстам, содержащим символы, отсутствующие в однобайтных кодировках (КОИ-8, windows-1251 и т. д.).

UTF-16

UTF-16 — один из способов кодирования символов (англ. code point) из Unicode в виде последовательности 16-битных слов (англ. code unit). Данная кодировка позволяет записывать символы Юникода в диапазонах U+0000..U+D7FF и U+E000..U+10FFFF (общим количеством 1 112 064), причем 2-байтные символы представляются как есть, а более длинные -с

помощью суррогатных пар (англ. surrogate pair), для которых и вырезан диапазон $D800_{16}..DFFF_{16}$.

В UTF-16 символы кодируются двухбайтовыми словами с использованием всех возможных диапазонов значений (от 0000_{16} до $FFFF_{16}$). При этом можно кодировать символы Unicode в диапазонах $0000_{16}..D7FF_{16}$ и $E000_{16}..10FFFF_{16}$. Исключенный отсюда диапазон $D800_{16}..DFFF_{16}$ используется как раз для кодирования так называемых суррогатных пар - символов, которые кодируются двумя 16-битными словами. Символы Unicode до $FFFF_{16}$ включительно (исключая диапазон для суррогатов) записываются как есть 16-битным словом. Символы же в диапазоне $10000_{16}..10FFFF_{16}$ (больше 16 бит) уже кодируются парой 16-битных слов. Для этого их код арифметически сдвигается до нуля (из него вычитается минимальное число 10000_{16}). В результате получится значение от нуля до $FFFFF_{16}$, которое занимает до 20 бит. Старшие 10 бит этого значения идут в лидирующее (первое) слово, а младшие 10 бит — в последующее (второе). При этом в обоих словах старшие 6 бит используются для обозначения суррогата. Биты с 11 по 15 имеют значения 11011_2, а 10-й бит содержит 0 у лидирующего слова и 1 - у последующего. В связи с этим можно легко определить к чему относится каждое слово.

UTF-32

UTF-32 — один из способов кодирования символов из Юникод, использующий для кодирования любого символа ровно 32 бита. Остальные кодировки, UTF-8 и UTF-16, используют для представления символов переменное число байт. Символ UTF-32 является прямым представлением его кодовой позиции (англ. code point).

Главное преимущество UTF-32 перед кодировками переменной длины заключается в том, что символы Юникод непосредственно индексируемы. Получение n-ой кодовой позиции является операцией, занимающей одинаковое время. Напротив, коды с переменной длиной требует последовательного доступа к n-ой кодовой позиции. Это делает замену символов в строках UTF-32 простой, для этого используется целое число в качестве индекса, как обычно делается для строк ASCII.

Главный недостаток UTF-32 — это неэффективное использование пространства, так как для хранения символа используется четыре байта. Символы, лежащие за пределами нулевой (базовой) плоскости кодового пространства редко используются в большинстве текстов. Поэтому

удвоение, в сравнении с UTF-16, занимаемого строками в UTF-32 пространства не оправдано.

Хотя использование неменяющегося числа байт на символ удобно, но не настолько, как кажется. Операция усечения строк реализуется легче в сравнении с UTF-8 и UTF-16. Но это не делает более быстрым нахождение конкретного смещения в строке, так как смещение может вычисляться и для кодировок фиксированного размера. Это не облегчает вычисление отображаемой ширины строки, за исключением ограниченного числа случаев, так как даже символ «фиксированной ширины» может быть получен комбинированием обычного символа с модифицирующим, который не имеет ширины. Например, буква «й» может быть получена из буквы «и» и диакритического знака «крючок над буквой». Сочетание таких знаков означает, что текстовые редакторы не могут рассматривать 32-битный код как единицу редактирования. Редакторы, которые ограничиваются работой с языками с письмом слева направо и составными символами (англ. Precomposed character), могут использовать символы фиксированного размера. Но такие редакторы вряд ли поддержат символы, лежащие за пределами нулевой (базовой) плоскости кодового пространства и вряд ли смогут работать одинаково хорошо с символами UTF-16.

6.3 Представление графической информации

Компьютерная графика — область деятельности человека, в которой компьютерные технологии используются для создания, обработки и хранения изображений.

В зависимости от области применения различают следующие виды компьютерной графики:

- Научная графика дает возможность проводить вычислительные эксперименты с наглядным представлением их результатов — графиков скалярных и векторных функций, заданных параметрических кривых и поверхностей.
- Деловая графика предназначена для наглядного представления различных показателей работы учреждений (схемы, диаграммы и т. д.).
- Конструкторская графика используется в работе инженеров-конструкторов, архитекторов (чертежи).
- Иллюстративная графика — произвольные рисунки.
- Художественная и рекламная графика — рекламные плакаты и ролики, компьютерные игры.

- Цифровая фотография и цифровая обработка изображений.
- Компьютерная анимация.

По способу представления изображения в ЭВМ, компьютерную графику разделяют на **векторную**, **растровую** и **фрактальную**.

6.3.1 Растровая графика. Под растровой графикой (bitmap, raster) понимают способ представления изображения в виде совокупности отдельных точек –**пикселей (*pixel*** - сокращение от ***picture element*** - элемент изображения) различных цветов или оттенков. Это наиболее простой способ представления изображения, ибо таким образом видит наш глаз.

Растровое изображение представляется в памяти ЭВМ в виде матрицы отдельных пикселей. Основными параметрами растрового изображения являются **разрешение** и **глубина** цвета.

Разрешение (resolution) - это степень детализации изображения, число пикселей, отводимых на единицу площади. Поскольку изображение можно рассматривать применительно к различным устройствам, то следует различать:

- разрешение оригинала;
- разрешение экранного изображения;
- разрешение печатного изображения.

Разрешение оригинала определяется при вводе изображения в компьютер и измеряется в точках на дюйм (***dots per inch*** - **dpi**). При этом количество **dpi** определяет не число точек в квадратном дюйме, а количество точек на одной его стороне. Например, 100 dpi означает, что в квадратный дюйм изображения покрывается растровой сеткой 100x100 и после сканирования, изображение соответствующее, квадратному дюйму будет состоять из 10 000 пикселей.

Для получения на экране изображения близкого к размеру оригинала обычно использует разрешения 72-75 dpi. Для вывода изображения в дальнейшем на печать и распознавания текста рекомендуется устанавливать разрешении 300-600 dpi. Если исходное изображение небольшого размера и его планируется увеличить и вывести на печать, то в этом случае разрешение оригинала лучше устанавливать 600-1200 dpi. Сканирование слайдов, негативных фотопленок и качественных материалов для полиграфии требует установки величины разрешения 1200 и более dpi.

Для измерения разрешения экранного изображения, кроме dpi, используют единица измерения **ppi** (***pixel per inch***). Размер пикселя, а

значит и разрешение экранного изображения, варьируется в зависимости от выбранного разрешения экрана (из диапазона стандартных значений), разрешения оригинала и масштаба отображения.

Глубина цвета (***color depth***) - это число бит, используемых для представления каждого пикселя изображения. С развитием вычислительных средств глубина цвета, хранимых в компьютере, изображений все время возрастала. Одним из первых распространенных стандартов мониторов являлся VGA, который поддерживал глубину цвета 8 бит для цветных изображений. Следующим шагом стало введение в компьютерах системы стандарта **HighColor**, который кодировал цвет с глубиной 16 бит, что позволяло получить 65536 цветов. Сейчас наиболее используемым является 24-битный **TrueColor**, позволяющий кодировать около 16,7 млн. цветов. Однако необходимо отметить, что существуют графические системы использующие глубину цвета более чем 24 бита на пиксель.

Поскольку линейные параметры можно выразить с помощью целых чисел, то растровое кодирование позволяет использовать двоичный код для представления графических данных.

Применение растровой графики:

- обработка цифровых фотографий;
- сканированных изображений;
- создание коллажей, эмблем, логотипов и т.п.

Растровые изображения чаще не создаются с помощью компьютера, а только обрабатываются.

Преимущества растровой графики:

- Растровая графика позволяет создать практически любой рисунок, вне зависимости от сложности, в отличие, например, от векторной, где невозможно точно передать эффект перехода от одного цвета к другому без потерь в размере файла;
- Распространённость -растровая графика используется сейчас практически везде: от маленьких значков до плакатов;
- Высокая скорость обработки сложных изображений, если не нужно масштабирование;
- Растровое представление изображения естественно для большинства устройств ввода-вывода графической информации, таких как мониторы (за исключением векторных устройств вывода), матричные и струйные принтеры, цифровые фотоаппараты, сканеры, а также сотовые телефоны.

Недостатки растровой графики:

- Большой размер файлов у простых изображений из большого количества точек;
- Невозможность идеального масштабирования;
- Невозможность вывода на печать на векторный графопостроитель.

Из-за этих недостатков для хранения простых рисунков рекомендуют вместо даже сжатой растровой графики использовать векторную графику.

Форматы растровых изображений

Существует множество различных форматов. Все они имеют разное сжатие, а некоторые и вовсе без него. Рассмотрим только самые часто используемые типы растровых картинок, с которыми вы можете столкнуться.

JPG или **JPEG** - самый распространенный формат и используемый. Обладает отличными параметрами сжатия за счет усреднения цвета. Большинство всех картинок в интернете существуют именно в этом формате.

PNG - распространен для интернета, очень качественный формат. Отлично подходит, необходимо сохранить высокое качество или, наоборот, использовать минимум цветов.

BMP - формат операционной системы Windows. Обычно вообще без сжатия, поэтому используется редко - потому, что передавать огромные файлы по интернету нет смысла и на жестком диске они занимают много места.

GIF - анимация. Максимум 256 цветов. Раньше пользовался большой популярностью, да и сейчас на многих ресурсах есть. С приходом HTML 5 практически потеряли всю надобность.

TIFF - часто используется для печати. Сохраняет исходные цвета в любом цветовом пространстве. Многие фотоаппараты делают снимки именно в этом формате.

RAW - наилучший формат для качественных снимков. Сохраняет изначальную картинку, которая попадает в объектив. В нем делают фото хорошие камеры. Такие снимки удобно редактировать. Недостатком является то, что занимают довольно много места.

6.3.2 Векторная графика - способ представления графических объектов и изображений (формат описания) в компьютерной графике, основанный на математическом описании элементарных геометрических объектов, обычно называемых примитивами, таких как: точки, линии, сплайны, кривые Безье, круги, окружности, эллипсы, многоугольники.

Объекты векторной графики описываются совокупностью координат, параметров и атрибутов.

При выводе на матричные устройства отображения (мониторы, механические устройства - принтеры, и пр.) векторная графика предварительно преобразуется в растровую графику. Преобразование производится программно или аппаратно средствами современных видеокарт. Для плоттеров и векторных мониторов преобразование из векторного в растровый формат не требуется, так как в таких устройствах примитивы строятся перемещением пера или электронного луча.

Математически векторная графика основана на аналитической геометрии. В векторной графике примитивы описываются совокупностью координат опорных точек, параметрами примитива, к которым относятся например, исходные координаты, коэффициенты масштабирования, повороты, коэффициенты растяжения по осям. К атрибутам относят цвет и тип цветовой заливки областей, толщину и цвет линий.

Например, для описания окружности радиуса R (графический примитив) необходимо задать следующие исходные данные:

- координаты центра окружности;
- значение радиуса;
- цвет и/или текстура заливки (если окружность построена для отображения круга);
- цвет и толщина линии контура в случае задания контура;
- прозрачность заливки и контура;
- порядок плана относительно других примитивов впереди, сзади (передний план, задний план). При отображении примитивы и объекты выводятся на устройство отображения последовательно и позднее выведенные объекты закрывают или накладываются на ранее выведенные объекты.

Координаты центра и радиус являются обязательными параметрами, остальные данные из описания окружности часто называют атрибутами и могут быть опущены. В этом случае при графическом выводе они заменяются атрибутами по умолчанию или текущими атрибутами.

Пример описания красного круга с координатами центра 85; 70, радиуса 10 с чёрным контуром толщиной 1 на языке XML (используемого в формате SVG):

```
<circle cx="85" cy="70" r="10" fill="#f00" stroke="#000" stroke-width="1"/>
```

В анимированной векторной графике при отображении на устройстве вывода координаты, атрибуты и параметры могут динамически

изменяться во времени по заданным функциям от времени, создавая движущееся изображение.

Основное применение векторной графики:

- Создание буклетов и разной печатной продукции;
- Создание логотипов компаний, визиток и т.п.;
- Создание новых, оригинальных шрифтов.

Преимущества векторной графики:

- Объём данных, занимаемый описанием примитивов не зависит от реальной линейной величины описываемого объекта, что позволяет описать сколько угодно большой объект файлом минимального размера;
- Так как информация об объекте хранится в числовой форме, можно бесконечно увеличивать изображение при выводе на устройство отображения;
- Описание объектов хранится в численном виде и может быть легко изменено и поэтому перемещение, масштабирование, вращение, заливка и т. д. не ухудшает качества изображения.

Принципиальные недостатки векторной графики:

- Не каждое изображение может быть компактно описана в векторном виде - для высокого подобия к оригинальному изображению может потребоваться описание очень большого количества примитивов, что требовательно к объёму памяти, занимаемой изображением и на время необходимое для преобразования его в растровый формат для графического вывода на растровые устройства отображения;
- Перевод векторной графики в растровое изображение алгоритмически достаточно прост. Но обратное преобразование, как правило, сложно - это преобразование называют трассировкой растрового изображения и зачастую требует значительных вычислительных мощностей и процессорного времени, и не всегда обеспечивает высокое качество полученного векторного изображения;
- Спецификации векторных форматов намного сложнее таковых для растровой графики;
- Преимущество векторной картинки - масштабируемость - пропадает, когда изображение в векторном формате отображается в растре малого разрешения.

Форматы векторной графики

На текущий момент создано свыше десятка форматов для работы с векторной графикой. Они разработаны производителями графических редакторов для собственных продуктов. Каждый формат обладает характерными плюсами и минусами – последние, в большинстве своем, касаются унификации и совместимости форматов. Наиболее используемые векторные форматы:

- **PDF** (Portable Document Format) Межплатформенный формат, созданный компанией Adobe. Максимально оптимизирован для создания файлов, содержащих как графическую информацию, так и текст. Поэтому относится к обычно применяемым для хранения логотипов в векторе, а также различной документации и презентаций. Обладает обширным набором шрифтов. Использует язык PostScript, с использованием инструментов которого возможно добавление анимационных вставок, ссылок и звуковых файлов. Большим плюсом формата является то, что графика в нем имеет компактные размеры и без искажений отображается на любых системах. Удобство работы и широкие возможности делают **формат PDF** наиболее популярным для работы с различной печатной продукцией.
- **SVG** (Scalable Vector Graphics) Разработанный на основе языка разметки XML, формат предусматривает работу с двумерной векторной графикой, используемой, например, при создании интернет-страниц. SVG-файлы могут редактироваться не только с помощью программ работы с графикой, но и некоторыми текстовыми редакторами. Документы в данном формате могут содержать как изображения и текст, так и анимацию. SVG прекрасно подойдет как для создания веб-иллюстраций, так и для редактирования файлов, поскольку даже в случае масштабирования графика в данном формате способна сохранить высокое качество.
- **AI** (Adobe Illustrator) Созданный специально для программы Adobe Illustrator данный формат обладает ограниченной совместимостью и с другими редакторами. Характеризуется стабильностью и высоким качеством изображения даже при значительном его масштабировании. AI – формат закрытого типа. Жестко привязан к версии Illustrator’a.
- **EPS** (Encapsulated PostScript) Применяется при подготовке полиграфических документов, поскольку удобен для печати больших объемов графической информации. Обеспечивает сохранение качества различных векторных логотипов при их редактировании

благодаря поддержке большого числа необходимых для обработки графики инструментов. Данный формат, разработанный Adobe, совместим со многими популярными редакторами графики.

- **CDR** (CorelDraw) Создан исключительно для редактора CorelDraw. Отсутствует совместимость практически со всеми иными приложениями, а также между макетами, созданными в разных версиях программы (файлы из новых версий не поддерживаются старыми). Файлы, созданные в данном формате способны обеспечить многослойность, прекрасно сохраняют параметры и спецэффекты объектов. Растровые и векторные изображения в CDR могут быть сжаты раздельно.

Как упоминалось выше, наиболее подходящими и часто используемыми при создании полиграфии и типовых иллюстраций являются форматы PDF и SVG.

6.3.3 Фрактальная графика. Фрактальная графика основана на математических вычислениях. Её базовым элементом является математическая формула. Никаких объектов в памяти компьютера не хранится и изображение строится исключительно по уравнениям либо системам уравнений. Таким способом строят как простейшие регулярные структуры, так и сложные иллюстрации, как, например, имитирующие природные ландшафты и трехмерные объекты.

Фрактальная графика является на сегодняшний день одним из самых быстро развивающихся и перспективных видов компьютерной графики. Основой построения фрактальной графики является **фрактал. Фрактал**- это структура самоподобных фрагментов. Это значит, что взяв небольшую часть фрактала можно получить информацию обо всем фрактале. Как повторяющиеся фоны на сайтах или узоры на коврах.

Следует отметить, что слово «фрактал» не является математическим термином и не имеет общепринятого строгого математического определения. Фигура является фракталом, если обладает следующими свойствами:

- имеет нетривиальную структуру во всех масштабах (для фрактала увеличение масштаба не ведет к упрощению структуры, поэтому на всех шкалах мы увидим одинаково сложную картину).
- является самоподобной или приближенно самоподобной.
- имеет дробную метрическую размерность или метрическую размерность, превосходящую топологическую.

Фракталы широко применяются в компьютерной графике для построения изображений природных объектов, таких как деревья, кусты, горные ландшафты, поверхности морей и так далее. Существует множество программ, служащих для генерации фрактальных изображений.

Фрактальная компьютерная графика позволяет создавать абстрактные композиции, где можно реализовать множество приёмов: горизонтали и вертикали, диагональные направления, симметрию и асимметрию. Фрактальное изображение можно сравнить, например, со сложной структурой кристалла, со снежинкой, элементы которой выстраивается в одну сложную композицию.

Для построения фрактала необходимо учитывать начальное состояние и описывающую его формулу - так называемое исходное множество, которое пропускается через некий механизм, вызывающий его отображение и добавляющий отображенное множество к исходному. (Этот процесс и называется итерацией). Таким образом, после нескольких подобных относительно простых операций получается сложное изображение.

В процессе получения фрактала важны два момента: исходное множество и механизм преобразования. В зависимости от алгоритма построения фракталы делятся на линейные и нелинейные.

Алгоритмы построения линейных фракталов определяются линейными функциями. В них самоподобие присутствует в простейшем варианте: любая часть повторяет целое.

Нелинейные фракталы задаются нелинейной функцией роста, то есть уравнениями в степени выше первой. В них самоподобие будет более сложным: любая часть является уже не точной, а деформированной копией целого.

Один из простейших примеров линейного фрактала - кривая Коха.

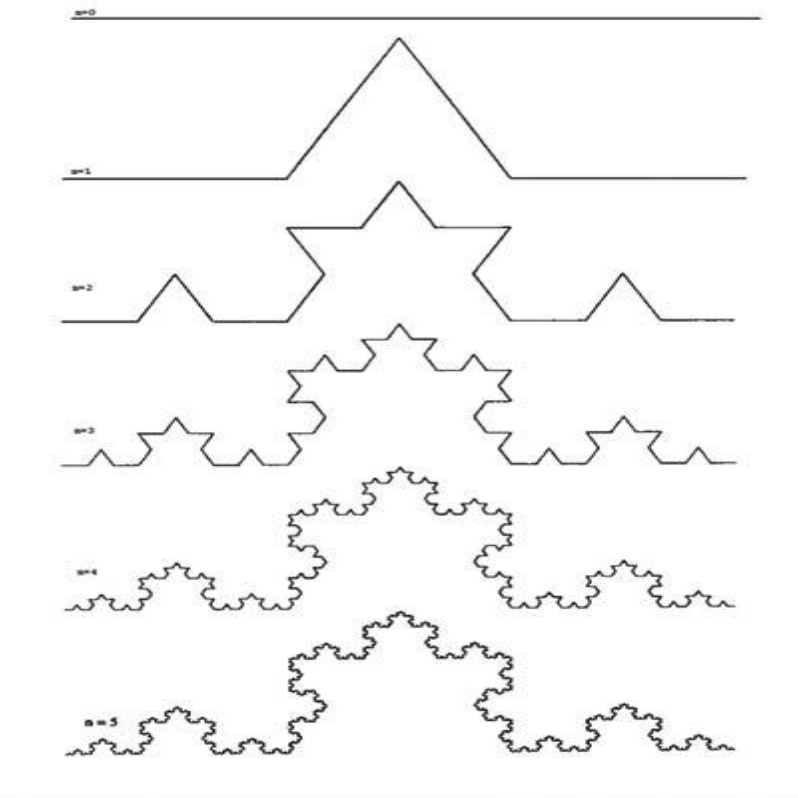


Рис. 6.3 Рекурсия кривой Коха

Существует простая рекурсивная процедура (получение самоподобных частей фрактала) формирования фрактальных кривых на плоскости. Зададим произвольную ломаную с конечным числом звеньев, называемую генератором. Далее заменим в ней каждый отрезок генератором (точнее, ломаной, подобной генератору). В получившейся ломаной вновь заменим каждый отрезок генератором. Продолжая до бесконечности, в пределе получим фрактальную кривую. На рис. 6.3 приведено несколько шагов этой процедуры для кривой Коха.

Одним из первых нелинейные фракталы описал французский математик Гастон Жюлиа еще в 1918 году. Но в его работе отсутствовали изображения исследованных им множеств и термин «фрактал».

В наше время компьютеры позволили получить изображения множеств Жюлиа (рис. 6.4), которые вместе с множествами Мандельброта (рис. 6.5) являются ныне самыми известными квадратичными фрактальными структурами.



Рис. 6.4 Изображение множества Жюлиа

Оба типа фракталов возникают в результате реализации на комплексной плоскости самого простого нелинейного алгоритма.

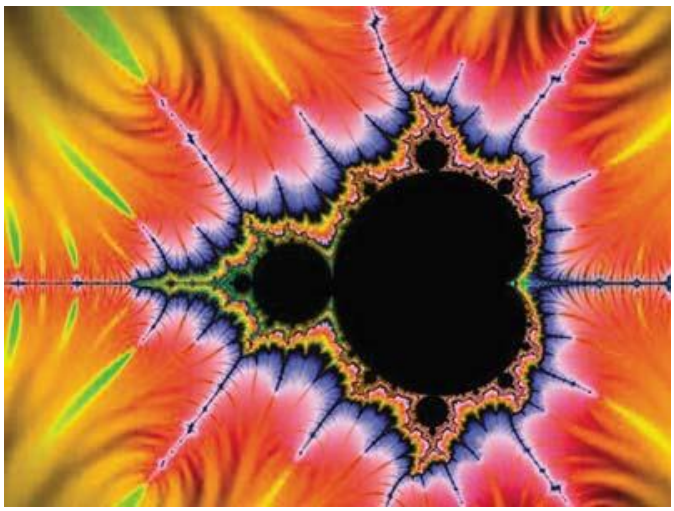


Рис. 6.5 Изображение множества Мальденброта

Здесь в основу метода построения изображений положен принцип наследования от так называемых родителей геометрических свойств объектов наследников. Построение фрактального рисунка осуществляется по какому либо алгоритму или путем автоматической генерации изображений при помощи вычислений по конкретным формулам. Изменения значений в алгоритмах или коэффициентов в формулах приводит к модификации этих изображений. Главным преимуществом

фрактальной графики является то, что в файле фрактального изображения сохраняются только алгоритмы и формулы.

Только изменив коэффициенты уравнения, можно получить совершенно иное изображение. Эта идея нашла применение в компьютерной графике благодаря компактности математического аппарата, необходимого для ее реализации. Так, с помощью нескольких математических коэффициентов можно задать линии и поверхности очень сложной формы.

В машинной графике фрактальная геометрия незаменима при генерации искусственных облаков, гор, поверхности моря. Фактически, благодаря фрактальной графике найден способ эффективной реализации сложных неевклидовых объектов, образы которых весьма похожи на природные. Многие природные объекты имеют фрактальные свойства, поэтому их легко создавать на компьютере с помощью фрактальной графики. Например, при разработке компьютерной игры нет нужды каждый раз заново рисовать лес, горы, облака и т.д. Эти объекты обладают самоподобием, и, следовательно, могут быть легко сгенерированы программными средствами на основе математических формул. Добавляя или изменяя некоторые параметры исходной формулы, можно добиться удивительного разнообразия получаемых природных объектов. Фракталы на экране компьютера - это узоры, построенные самим ПК по заданной программе.

Следует отметить, что фрактальная графика - одно из самых необычных и перспективных направлений в компьютерной графике. Результаты, которые можно получить с ее помощью, поражают воображение даже самых искушенных ценителей компьютерного искусства. Так, изображения, создаваемые с помощью программ-фракталогенераторов, порой содержат совершенно фантастические и необычные картины. И наоборот, с помощью фрактальной графики можно с удивительной точностью изобразить то, что мы видим в окружающем нас мире. Мир фракталов удивителен!

Помимо фрактальной живописи существуют фрактальные анимация и музыка.

Применение фрактальной графики. Область применения фрактальной графики постоянно расширяется. На данный момент можно отметить следующие области:

- Компьютерная графика. Реалистичное изображение рельефов и природных объектов.
- Использование при создании компьютерных игр.

- Анализ фондовых рынков. Использование фракталов отметить повторения, которые впоследствии сыграют трейдерам на руку.
- Естественные науки. В физике, например, с помощью фрактальной графики моделируются нелинейные процессы. В биологии она описывает строение кровеносной системы.
- Сжатие изображений, чтобы уменьшить объем информации.

Достоинства фрактальной графики:

- Небольшой размер при масштабном рисунке.
- Нет конца масштабированию, сложность картинки можно увеличивать бесконечно.
- Нет другого такого же инструмента, который позволит создавать сложные фигуры.
- Реалистичность.
- Простота в создании работ.

Недостатки фрактальной графики:

- Чем длиннее количество повторений, тем больше загружается процессор. Соответственно, только качественное компьютерное оборудование способно справиться с построением сложных изображений.
- Существуют ограничения в исходных математических фигурах. Некоторые изображения создать посредством фракталов не удастся.

6.3.4 Цветовые модели компьютерной графики

Цветовая модель - средство представления цвета посредством его разложения на простые составляющие и его описания в виде набора чисел (математическая модель).

Цветовых моделей много, в них заложены разные принципы работы с цветами и разные возможности для их отображения. Все используемые в настоящее время цветовые модели можно условно классифицировать следующим образом:

1. Монохромные:
   - двухградационные (черно-белые).
   - полутоновые (с оттенками серых цветов).
2. Цветные:
   - индексные - каждому цвету в цветовой таблице ставится в соответствие индекс.
   - Полноцветные:
     - аддитивные (**RGB**) — основанные на сложении цветов;
     - субтрактивные (**CMY, CMYK**) — основанные на вычитании

цветов;

- перцепционные (**HSV, HSB, HLS, Lab**, и т. д.) — основанные на восприятии цвета.

Цветовая модель RGB - самый популярный способ представления графики, который подходит для описания цветов, видимых на мониторе, телевизоре, видеопроекторе, а также создаваемых при сканировании изображений.

Для кодирования цветных графических изображений применяется ***принцип декомпозиции*** произвольного цвета на основные составляющие. В качестве таких составляющих используют ***три основных цвета***: красный (Red, R), зеленый (Green, G) и синий (Blue, B). Такая система кодирования называется модель **RGB** (по первым буквам названий основных цветов). Остальные цвета получаются сочетанием базовых. Цвета такого типа называются аддитивными, поскольку при сложении (смешивании) двух лучей основных цветов результат становится светлее. На рис. 6.6 показано, какие цвета получаются при сложении основных



Рис. 6.6

Графические файлы, в которых применяется цветовая модель **RGB**, представляют каждый пиксель в виде ***цветового триплета*** -трех числовых величин (R, G, B), соответствующих интенсивностям красного, зеленого и синего цветов. Цветовая модель RGB можно представить в виде куба (рис. 6.7), вершины которого располагаются на осях, соответствующих красному, зеленому и синему цветам.

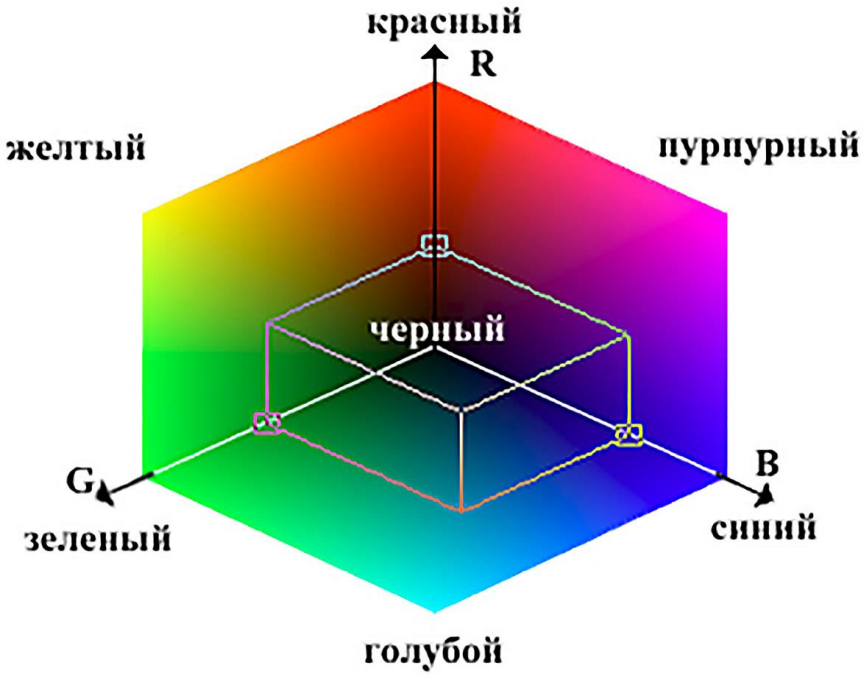


Рис. 6.7

Если для кодирования яркости каждой из основных составляющих использовать по 256 значений (восемь двоичных разрядов), как это принято для полутоновых черно-белых изображений, то на кодирование цвета одной точки надо затратить 24 разряда. При этом система кодирования обеспечивает однозначное определение 16,5 млн. различных цветов, что близко к чувствительности человеческого глаза.

Для 24-битового цвета триплетом (0, 0, 0) обычно представляется черный цвет, а триплетом (255, 255, 255) — белый. Если все три величины *RGB* имеют одинаковые значения, например, (63, 63, 63), (127, 127, 127) или (191, 191, 191), то результирующим будет один из оттенков серого цвета.

Достаточно часто используется подвид данного цветового пространства - модель **RGBA**, где A (Alpha) означает прозрачность цвета, т.е. насколько сквозь данный цвет просвечивает фон.

Кроме **RGB**, другими популярными системами кодирования цветных изображений являются **CMY** и **HSB**.

Цветовая модель CMY (Cyan-Magenta-Yellow - голубой-пурпурный-желтый) - цветовая модель, применяемая для получения цветных изображений на белой поверхности. Эта модель используется в большинстве устройств вывода, таких как лазерные и струйные принтеры, типографские станки.

При печати на бумагу наносится краска - материал, который поглощает и отражает цветовые волны различной длины. Таким образом, краска выступает в роли фильтра, пропускающего строго определенные лучи отраженного цвета, вычитая все остальные.

Цвета этой модели получаются в результате вычитания из белого базовых цветов модели RGB. Поэтому их называют субтрактивными.
Базовыми для **CMY** являются следующие цвета:

- голубой (Cyan) — белый минус красный (Red);
- пурпурный (Magenta) — белый минус зеленый (Green);
- желтый (Yellow) — белый минус синий (Blue).

На практике получить идеальный черный цвет без дорогостоящих красителей в модели **CMY** весьма сложно. Существует более практичный вариант **CMY**-модель **CMYK**, в которой символ **K** означает черный цвет, и который является ключевым (Key) в процессе цветной печати. Дело в том, что реальные краски имеют примеси, поэтому их цвет не соответствует в точности теоретически рассчитанным голубому, пурпурному и желтому. Смешение трех основных красок, которые должны давать черный цвет, дает вместо этого неопределенный грязно--коричневый. Поэтому в число основных полиграфических красок и внесена черная.

Модель **CMYK** часто называют четырехцветной, а результат ее применения - четырехцветной печатью.

На рис. 6.8 представлена схема, из которой видно, какие цвета получаются при смешении базовых в **CMYK**.

Как правило, четыре цветовые составляющие **CMYK** задаются в процентах в диапазоне от 0 до 100. Следует отметить, что краски модели **CMYK** не являются столь чистыми, как цвета модели . Этим объясняется небольшое несоответствие базовых цветов. Согласно схеме, представленной на рис. 6.8, при максимальной яркости должны получаться следующие комбинации цветов:

- смешение пурпурного (M) и желтого (Y) должно давать красный цвет (R) (255, 0, 0);
- смешение желтого (Y) и голубого (C) должно давать зеленый цвет (G) (0, 255, 0);
- смешение пурпурного (M) и голубого (C) должно давать синий цвет (B) (0, 0, 255).



Рис. 6.8

Режим, когда для кодирования цвета каждой точки используется 32 двоичных разряда, также называют ***полноцветным*** (**True Color**). Кодирование цветной графики 16-разрядными двоичными числами называют **режимом High Color**.

Одним из недостатков растровых методов является трудность пропорционального изменения размеров изображения до произвольно выбранного значения.

Цветовые модели **RGB** и **CMYK** являются аппаратно-зависимыми. Для модели **RGB** значения базовых цветов определяются качеством люминофора у ЭЛТ или характеристиками ламп подсветки и цветовых фильтров панели у ЖК-мониторов. Если обратиться к модели **CMYK**, то значения базовых цветов определяются реальными типографскими красками, особенностями печатного процесса и носителя. Таким образом, одинаковое изображение может на различной аппаратуре выглядеть по-разному.

Как уже отмечалось, **RGB** является наиболее популярной и часто применяемой моделью для представления цветных изображений. В большинстве случаев изображения подготавливаются для демонстрации через монитор или проектор и для печати на цветных настольных принтерах. Во всех этих случаях необходимо использовать модель **RGB.**

Следует отметить, что несмотря на то, что в цветных принтерах используются чернила цветовой модели **CMYK**, чаще всего изображения,

подготавливаемые для печати, необходимо преобразовать в модель **RGB**. Но распечатанное изображение будет выглядеть немного темнее, чем на мониторе, поэтому перед печатью его необходимо осветлить. Величина осветления для каждого принтера определяется опытным путем.

Модель **CMYK** необходимо применять в одном случае - если изображение готовится к печати на типографском станке. Более того, следует учесть, что модель CMYK не содержит столь же большого числа цветов, как RGB, поэтому в результате преобразования из **RGB** в **CMYK** изображение может утратить ряд оттенков, которые вряд ли получится восстановить обратным преобразованием. Поэтому старайтесь выполнять преобразование изображения в модель **CMYK** на конечном этапе работы с ним.

Цветовая модель HSB

Модель HSB упрощает работу с цветами, так как в ее основе лежит принцип восприятия цвета человеческим глазом. Любой цвет определяется своим цветовым тоном (Hue) - собственно цветом, насыщенностью (Saturation) -процентом добавления к цвету белой краски и яркостью (Brightness) -процентом добавления черной краски. На рис. 6.9 показано графическое представление модели HSB.

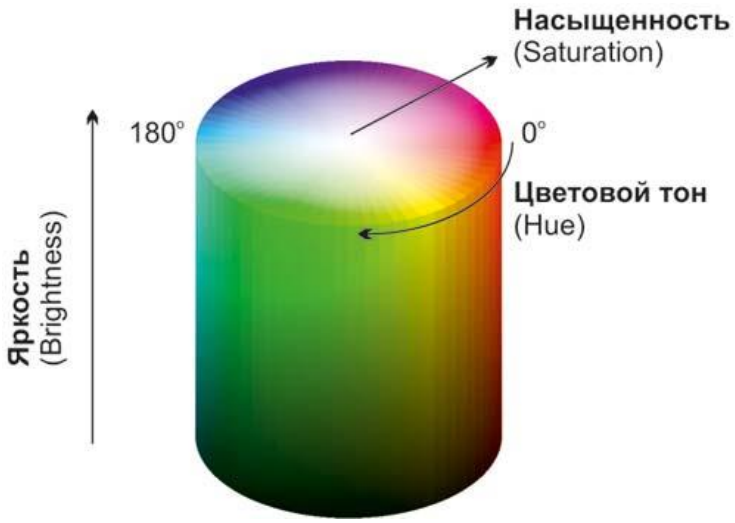


Рис. 6.9

Спектральные цвета, или цветовые тона, располагаются по краю цветового круга и характеризуются положением на нем, которое определяется величиной угла в диапазоне от 0 до 360°. Эти цвета обладают максимальной (100%) насыщенностью (S) и яркостью (B). Насыщенность

изменяется по радиусу круга от 0 (в центре) до 100% (на краях). При значении насыщенности 0% любой цвет становится белым.

Яркость — параметр, определяющий освещенность или затемненность. Все цвета цветового круга имеют максимальную яркость (100%) независимо от тона. Уменьшение яркости цвета означает его затемнение. Для отображения этого процесса на модели добавляется новая координата, направленная вниз, на которой откладываются значения яркости от 100 до 0%. В результате получается цилиндр, образованный из серии кругов с уменьшающейся яркостью, нижний слой - черный.

В модели HSB любой цвет получается из спектрального добавлением определенного процента белой и черной красок. Поэтому HSB — очень простая в понимании модель, которую используют маляры и профессиональные художники. У них обычно есть несколько основных красок, а все другие получаются добавлением к ним черной или белой. Однако при смешивании художниками красок, полученных на основе базовых, цвет выходит за рамки модели HSB

Цветовая модель Lab

Модель Lab основана на следующих трех параметрах: ***L*** — яркость (Lightness) и два хроматических компонента - ***a*** и ***b*.**

Параметр *a* изменяется от темно-зеленого через серый до пурпурного цвета. Параметр *b* содержит цвета от синего через серый до желтого (См. рис.6.10). Оба компонента меняются от –128 до 127, а параметр *L* — от 0 до 100. Нулевое значение цветовых компонентов при яркости 50 соответствует серому цвету. При значении яркости 100 получается белый цвет, при 0 — черный.

Понятия яркости в моделях **Lab** и **HSB** нетождественны. Как и в **RGB**, смешение цветов из шкал a и b позволяет получить более яркие цвета. Уменьшить яркость результирующего цвета можно за счет параметра L.

В модели **Lab** яркость полностью отделена от изображения, поэтому в некоторых случаях эту модель удобно использовать для перекраски фрагментов и повышения насыщенности изображения, влияя только на цветовые составляющие a и b. Также возможна регулировка контраста, резкости и других тоновых характеристик изображения за счет изменения параметра яркости L.

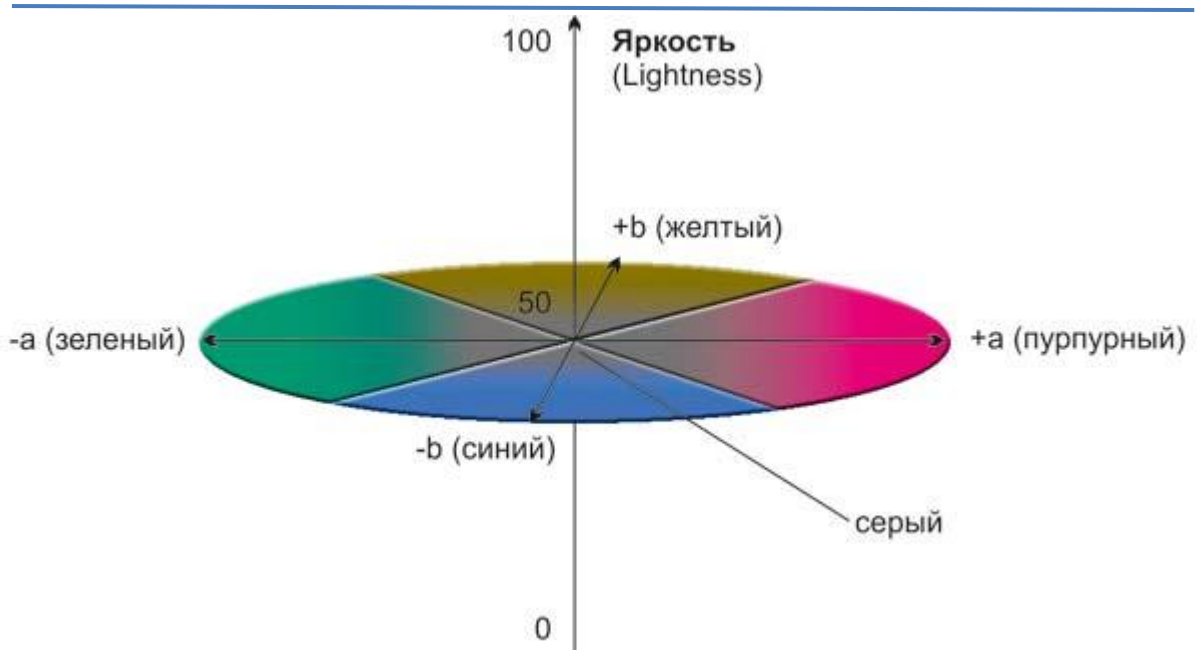


Рис. 6.10

Цветовой охват модели **Lab** шире, чем у **RGB**, поэтому каждое повторное преобразование из одной модели в другую практически безопасно. Более того, можно перевести изображение в режим **Lab,** выполнить коррекцию в нем, а затем безболезненно перевести результат обратно в модель **RGB.**

Модель **Lab** аппаратно независима, служит ядром системы управления цвета в графическом редакторе Photoshop и применяется в скрытом виде при каждом преобразовании цветовых моделей как промежуточная. Ее цветовой диапазон покрывает диапазоны **RGB** и **CMYK.**

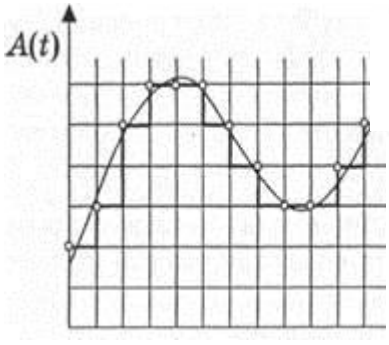
6.4 Представление звуковой информации

Из физики известно, что звук определяется как колебание среды. Наиболее часто средой передачи звуковых колебаний является воздух, но звук прекрасно распространяется и в более плотных средах. Следует отметить, что диапазон частот звуковых волн, воспринимаемых человеком, лежит в пределах примерно от нескольких десятков герц до чуть более 20 кГц. Благодаря роли звуковых сигналов в практической жизни человека, процессы генерации и закономерности распространения звука изучены достаточно хорошо. Чаще всего звуковые колебания преобразуются в электрические с помощью специальных устройств для дальнейшего их использования в различных областях.

Важную роль в анализе звуковых (или полученных из них электрических) колебаний играет также спектральный анализ, т.е. нахождение распределения интенсивности различных частот в исходном сигнале. Математической основой такой процедуры служит разложение

изучаемой функции в ряд по гармоническим функциям (синусам или косинусам) – так называемый Фурье-анализ. Полученные в результате обработки спектры также обычно представляются графически в координатах частота (абсцисса) – интенсивность (ордината). Чтобы представить себе, как выглядит спектр звукового сигнала, достаточно взглянуть на информационный дисплей современного высококачественного аудиокомплекса.

Звуковые сигналы в окружающем нас мире необычайно разнообразны. Для их записи с целью последующего воспроизведения необходимо как можно точней сохранить форму кривой зависимости интенсивности звука от времени. При этом возникает одна очень важная и принципиальная трудность: звуковой сигнал непрерывен, а компьютер способен хранить в памяти только дискретные величины. Отсюда следует, что для обработки, хранения и воспроизведения *звуковой информации* средствами вычислительной техники она должна быть «***оцифрована***», т.е. из аналоговой непрерывной формы переведена в цифровую дискретную, т.е. закодирована. Для этого необходимо осуществить ее ***дискретизацию***. Данную функцию выполняет специальный блок, входящий в состав звуковой карты, который называется ***аналого-цифровой преобразователь – АЦП***. Делается это, например, так: измеряется напряжение через равные промежутки времени и полученные значения записываются в память компьютера и при этом непрерывная зависимость амплитуды сигнала от времени *A(t)* заменяется ступенчатой (Рис.6.11).



Значения оцифрованного сигнала при глубине n =3

0010	0110	0011
0011	0110	0011
0101	0101	0100
0110	0100	0101

Рис.6.11

Точность округления зависит от выбранного количества (2^n) уровней квантования, которое, в свою очередь, зависит от количества бит (n), отведенных для записи значения амплитуды. Число n называют ***глубиной квантования*** или разрядностью квантования (т.е. бит, в каждом слове), а полученные в результате округления значений амплитуды числа - отсчетами или ***сэмплами*** (от англ. “ sample” – “замер”). Этот способ

оцифровки сигнала - дискретизация сигнала во времени в совокупности с методом однородного квантования - называется ***импульсно-кодовой модуляцией***, ***ИКМ*** или ***PCM*** (англ. ***Pulse Code Modulation***). Рис. 6.12

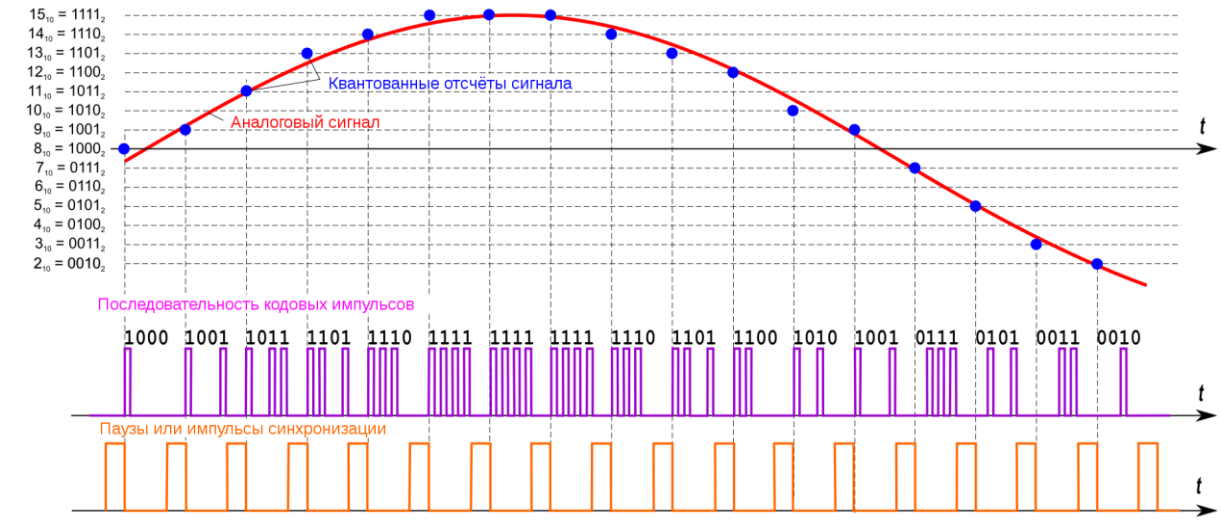


Рис. 6.12

При импульсно-кодовой модуляции аналоговый передаваемый сигнал преобразуется в цифровую форму посредством трёх операций: дискретизации по времени, квантования по амплитуде и кодирования (Рис. 6.12). АЦП через равные промежутки времени измеряет амплитуду аналогового сигнала получает мгновенные значения или отсчёты сигнала и преобразует отсчёты в двоичные слова. Мгновенное измеренное значение (отсчёт) аналогового сигнала по уровням квантуется (т.е. округляется от ближайшего целого). Номер уровня кодируется двоичными словами длиной *n* бит.

Следует отметить, что если известна глубина кодирования, то количество уровней громкости цифрового звука определяется как

$$\boldsymbol{N = 2^n}. \qquad (6.1)$$

Объем требуемой памяти для хранения звукового файла длительностью $\boldsymbol{t}$ секунд, оцифрованным с частотой дискретизации $\boldsymbol{f}$ (Гц) и глубиной кодирования $\boldsymbol{n}$ бит рассчитывается по формуле

$$\boldsymbol{M = t \cdot f \cdot n} \text{ (бит)}. \qquad (6.2)$$

При двухканальной записи (стерео) объем памяти, необходимый для хранения звукового файла, рассчитывается по формуле (6.2) умноженной на 2, а при четырехканальной записи (квадро) – умноженной на 4.

Существуют и другие способы оцифровки звукового сигнала, как, например,:

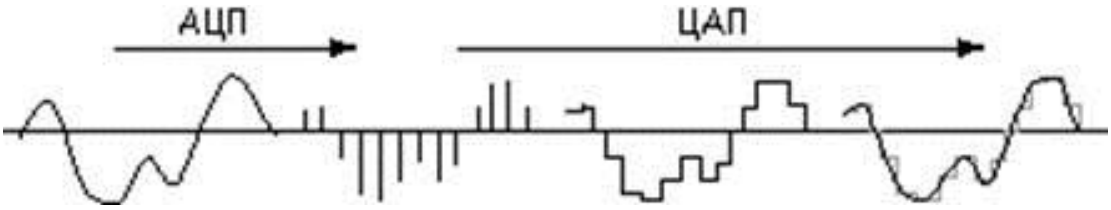
Способ неоднородного квантования, который предусматривает разбиение амплитудной шкалы на уровни по логарифмическому закону.

Такой способ квантования называют ***логарифмическим квантованием***. При использовании логарифмической амплитудной шкалы, в области слабой амплитуды оказывается большее число уровней квантования, чем в области сильной амплитуды (при этом, общее число уровней квантования остается таким же, как и в случае однородного квантования). Аналогово-цифровое преобразование, основанное на применении метода неоднородного квантования, называется ***неоднородной импульсно-кодовой модуляцией*** - неоднородной ***ИКМ*** (***Nonuniform PCM***).

Альтернативным способом аналогово-цифрового преобразования является ***разностная импульсно-кодовая модуляция*** – разностная ***ИКМ*** (***Differential PCM – DPCM***), при которой сигнал кодируется в виде разности между текущим и предыдущим измеренными значениями. Для звуковых данных такой тип модуляции уменьшает требуемое количество бит на отсчёт примерно на 25 %.

Качество полученного цифрового звука зависит от количества измерений уровня громкости звука в единицу времени, т. е. ***частоты дискретизации*** - количество измерений громкости звука за одну единицу времени (секунду). Чем большее количество измерений производится за одну секунду (т.е. чем больше частота дискретизации), тем точнее "лесенка" цифрового звукового сигнала повторяет кривую диалогового сигнала. Ну и как видно из изложенного, второй составляющей качества оцифрованного звука является глубина квантования, т.е. количество двоичных разрядов, используемых для кодирования динамического диапазона звуковых сигналов. Именно эти величины и являются характеристиками используемого аудиоадаптера. Отметим, что разрядность регистра аудиоадаптера определяет значение глубины квантования.

Для воспроизведения оцифрованного звукового файла необходимо выполнить обратное преобразование, т.е. из дискретной цифровой формы представления сигнала получить аналоговую, непрерывную**.** Для этого служит ***цифро-аналоговый преобразователь – ЦАП***.



Процесс восстановления первоначального аналогового сигнала по имеющимся дискретным данным нетривиален, поскольку никакой информации о форме сигнала между соседними отсчетами не

сохранилось. В разных звуковых картах для восстановления звукового сигнала могут использоваться различные способы.

Наиболее наглядный и понятный из них состоит в том, что по имеющимся точкам рассчитывается степенная функция, проходящая через заданные точки, которая и принимается в качестве формы аналогового сигнала. Чтобы понять, как это делается, возьмем, например, интерполяцию параболой $I = at^2 + bt + c$ по трем заданным точкам. Подставив в эту формулу известные значения времени и приравняв их к сохраненным в файле значениям интенсивности звука I, получим три линейных уравнения с тремя неизвестными a, b и c. Качественный вид результата представлен на рисунке 6.13

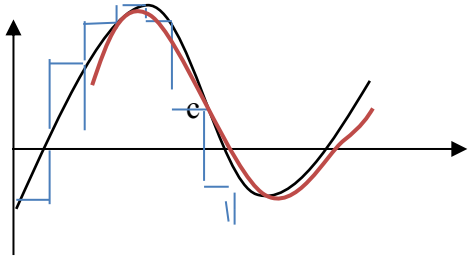


Рис. 6.13

Важную роль при оценке качества аудиоформатов и, соответственно, качества звука является такой параметр как ***битрейт***, который показывает, сколько бит или килобит необходимо для записи одной секунды звука. Малые значения битрейта – низкокачественное звучание, большие – звук высшего качества.

6.2 Форматы звуковых файлов

Звуковые файлы имеют несколько форматов представления. Основополагающим для всех звуковых файлов является формат .***PCM***, но файлы именно с таким расширением встречаются довольно редко.

6.2.1.Три типа аудио форматов

- Без сжатия.
- Сжатие без потерь.
- Сжатие с потерями.

Форматы записи звука без сжатия демонстрируют все свое лучшее только на высококачественном, профессиональном звуковоспроизводящем оборудовании.

Например, обычный планшет или смартфон не в состоянии воспроизвести прекрасное и чистое звучание музыкального произведения просто потому, что программно-аппаратные средства, динамики либо

наушники имеют характеристики низкокачественного воспроизведения звуковых файлов.

С другой стороны, если запустить звуковой файл формата ***MP3*** через профессиональные стереосистемы и усилители, то в динамиках будет слышан и шум, и скрежет, что использование этого оборудования окажется совершенно бессмысленным.

Имеет место рейтинг звуковых файлов по типу звуковоспроизводящего оборудования:

1. Для профессионального оборудования – форматы аудио без сжатия.
2. Для полу-профессионального оборудования - форматы аудио со сжатием, но без потерь.
3. Для бюджетного оборудования – форматы аудио со сжатие и с потерями.

Форматы аудио для высокого качества звучания

А) **несжатые форматы**:

Формат PCM - наиболее распространенный формат записи звука, используемый на CD и DVD дисках. Оригинальный аналоговый звук оцифровывается “как есть”, без каких-либо изменений. При условии качественных динамиков звучание почти один-в-один с живым исполнением.

Формат WAV разработан в 1991 году и является наиболее простым форматом хранения дискретных данных. ***WAV***-один из типов файлов семейства ***RIFF*** (***Resource Interchange File Format***- уникальная система хранения любых структурированных данных). Помимо обычных дискретных значений: битности, количества каналов, количества значений уровней громкости, в ***WAV*** может быть указано еще множество параметров: метки позиций для синхронизации, общее количество дискретных значений, порядок воспроизведения различных частей звукового файла, а также выделено место для размещения текстовой информации, и поэтому многие считают ***WAV*** несжатым форматом. Но на самом деле, как видно, это контейнер и там могут содержаться в том числе и сжатые файлы.

В большинстве случаев ***WAV*** содержит несжатый звук в формате ***PCM***. Поэтому и качество высокое. Но и на одну минуту записи тратится около 32MB памяти. Следует отметить, что формат ***WAV*** имеет достаточно хорошую совместимость с Windows.

Формат *AIFF* является аналогом *WAV* от разработчика Apple. Тоже контейнер и тоже содержит чаще всего звук в формате PCM. Хорошая совместимость с Windows.

Сжатые аудио форматы с потерями

Формат MP3 – Самый популярный формат сжатия на сегодняшний день (формат кодирования звуковой дорожки MPEG) - лицензируемый формат файла для хранения аудио-информации. Формат ***MP3*** (***MPEG Layer 3***) был разработан, после ряда промежуточных форматов, в Германии. Имеет небольшой вес (коэффициент сжатия 10-12), отличную совместимость с разными программами и аудиоустройствами. Но наряду с этим формат обеспечивает не совсем высокое качество воспроизведения и имеет ограниченные технические возможности.

Формат MP3 Pro был создан компанией Coding Technologies совместно с Thomson Multimedia в 2001 году. В его основе лежит ***MP3***. В нем используется технология ***SBR*** (***Spectral Band Replication***), за счет чего обеспечивается несколько лучшее качество воспроизведения. Формат ***MP3 Pro*** применяется больше для трансляций в интернете и демонстраций фрагментов новых музыкальных композиций.

Формат AAC является запатентованным форматом с меньшим, чем у MP3 потерей качества и с большими возможностями кодирования. Имеет улучшенные звуковые характеристики и большую степень сжатия. Применяется на Android, iOS, iTunes, YouTube, Nintendo и последних версиях PlayStation.

Формат OGG В общем, это не формат, а контейнер и, по сути, название OGG ни о чем не говорит в отношении содержащегося в нем звука. Однако чаше всего содержит кодек ***Vorbis.*** (кодек-устройство).

Vorbis — свободный формат сжатия звука с потерями, появившийся в 2002 году, который по принципам действия близок к ***MP3*** и ему подобным.

Для хранения аудиоданных в формате ***Vorbis*** чаще всего применяется медиаконтейнер ***Ogg***, такой файл обычно имеет расширение ***.ogg*** и называется двойным именем ***Ogg Vorbis***. Формат распространён существенно меньше, чем *MP3*. По всевозможным оценкам является вторым по популярности форматом сжатия звука с потерями. Широко используется в компьютерных играх и в файлообменных сетях для передачи музыкальных произведений.

Формат не ограничивает пользователя только двумя аудиоканалами (стерео - левый и правый), а поддерживает до 255 отдельных каналов с

частотой дискретизации до 192 кГц и разрядностью до 32 бит (чего не позволяет ни один другой формат сжатия с потерями), поэтому ***Vorbis*** великолепно подходит для кодирования 6-канального звука ***DVD-Audio***.

Несмотря на то что это самый молодой формат из всех конкурентов *MP3*, ***Ogg Vorbis*** имеет полную поддержку на всех известных платформах (Microsoft Windows, GNU/Linux, MacOS, PocketPC, Palm, Symbian, DOS, FreeBSD, BeOS и др.), а также большое количество аппаратных реализаций. Популярность на сегодняшний день значительно превосходит все альтернативные решения.

Формат WMA (***Windows Media Audio***) - лицензируемый формат файла, разработанный компанией Microsoft для хранения и трансляции аудио-информации. Номинально формат ***WMA*** характеризуется хорошей способностью сжатия, что позволяет ему «обходить» формат *MP3* и конкурировать по параметрам с форматами *Ogg Vorbis* и *AAC*. Но как было показано независимыми тестами, а также при субъективной оценке качество форматов все таки не является однозначно эквивалентным, а преимущество даже перед *MP3* однозначным. *Формат WMA* широкого применения не получил, но если есть возможность использовать *WMA* вместо *MP3*, то рекомендуется использовать *WMA* из-за его экономичности и более качественного воспроизведения.

Сжатые форматы аудио без потерь

Рекомендуется использовать для высококачественного воспроизведения звуковых файлов с использованием соответствующей аппаратуры. Но недостатком таких форматов является то, что размеры файлов одинаковой по длительности записи будут примерно в два или три раза больше. Следует однако заметить, что декларируемое сжатие без потерь не является абсолютным, т.к. минимальные, пусть и заметные только для “звуковых экспертов” потери все же есть. И кроме этого, не следует путать сжатие без потерь с ***Hi-Fi*** аудиозаписями.

Формат FLAC (***Free Lossless Audio Codec*** - свободный аудиокодек без потерь) - популярный свободный кодек для сжатия аудио. В отличие от кодеков с потерями *Ogg Vorbis*, *MP3* и *AAC*, не удаляет никакой информации из аудиопотока и подходит как для ежедневного прослушивания, так и для архивирования аудиоколлекции. На сегодня формат *FLAC* поддерживается многими аудиоприложениями.

FLAC является членом семейства кодеков, разрабатываемых Xiph.Org. К слову, в него же входит известный *Ogg Vorbis* – один из лучших lossy-алгоритмов сжатия музыки. В качестве контейнера для

аудиоданных используется, разумеется, OGG (файлы с расширением ***.ogg***) и ещё один open-source контейнер – Matroska (файлы с расширением ***.mka***).

Формат WMA Lossless - улучшенная версия вышеупомянутого WMA. Незначительно уступает *FLAC* и *ALAC*. Обладает существенным преимуществом перед *ALAC*,так как WMA хорошо поддерживается на Windows и Mac. Однако широкого распространения не имеет, поэтому если у вас в пользовании много разных устройств – вероятна несовместимость.

Конечно, это далеко не полный перечень применяемых форматов звуковых файлов.



Лекция 7.
Логические основы ЭВМ.
7. Функции алгебры логики и их основные свойства.

7.1 Основные определения

Согласно принципу фон-Неймана, реализация алгоритма обработки информации в ЭВМ происходит путем последовательного выполнения элементарных операций, к которым относятся, например, сложение, сдвиг, установка, прием, преобразование и много других. Для выполнения каждой из этих операций разработаны специальные электронные узлы - регистры, счетчики, сумматоры, преобразователи кодов, контроллеры и т.д.

Математический аппарат, описывающий действия дискретных устройств, к которым относится и ЭВМ, основывается на ***алгебре логики***, автором которой считается английский математик Дж. Буль (1815 – 1864г.). В практических целях первым применил его американский ученый К. Шеннон в 1938 г. при исследовании электрических цепей с контактными выключателями. Алгебра логики оперирует двоичными переменными (***logic values***), которые условно обозначают, как 0 и 1 и называют двоичной цифрой (***binary digit***) или битом (***bit***). В ее основе лежит понятие переключательной, или ***функции алгебры логики*** (***булевой функции***) вида $f(x_{n-1}, x_{n-2}, \cdots, x_1, x_0)$[3] относительно n аргументов $x_{n-1}, x_{n-2}, \cdots, x_1, x_0$, которая также может принимать лишь два значения - 0 и 1.

Основным понятием алгебры логики является – ***высказывание,*** на основе которого формируются ряд основных определений алгебры логики:

1. Определение 1. ***Высказывание*** - некоторое предложение, о котором можно утверждать, что оно ***истинно*** или ***ложно***. Любое высказывание можно обозначить символом, например, x и считать, что x=1, если высказывание истинно, x=0 если высказывание ложно.
2. Определение 2. ***Высказывание абсолютно истинно***, если соответствующая ей логическая величина x принимает единичное значение при любых условиях.
3. Определение 3. ***Высказывание абсолютно ложно***, если соответствующая ей логическая переменная x принимает нулевое значение при всех условиях.

[3] Часто функцию алгебры логики от n логических переменных записывают как $f(x_1, x_2, \cdots x_n)$. В дальнейшем в данном пособии будет использоваться такая запись

4. Определение 4. ***Логическая переменная*** – переменная величина x, которая может принимать только два числовых значения: $x = \{0, 1\}$. Следует отметить, что довольно часто значение логического нуля заменяют выражением «***ложь***», а значение логической единицы выражением «***истина***». Это является следствием того, что алгебра логики является булевой, интерпретирующей алгебру высказываний.
5. Определение 5. ***Функцией алгебры логики*** **(ФАЛ)** **(*Логической функцией*)** называется функция $f(x_1, x_2, \cdots x_n)$ принимающая значение, равное нулю или единице на наборе логических переменных $(x_1, x_2, \cdots x_n)$.

Если имеется n логических переменных, каждая из которых принимает значение 0 или 1, то набор можно рассматривать как некоторое двоичное число (двоичный набор) и тогда общее количество двоичных наборов равно

$$N = 2^n. \qquad (7.1)$$

Если каждому двоичному набору по определенному правилу поставлено в соответствие значение 0 или 1, то говорят, что задана функция алгебры логики.

Определение 6. Если две ФАЛ $f_1(x_1, x_2, \cdots x_n)$ и $f_2(x_1, x_2, \cdots x_n)$ принимают на всех возможных наборах значений аргументов одинаковые значения, то функции f_1 и f_2 называются равными, т.е.:

$$f_1(x_1, x_2, \cdots x_n) = f_2(x_1, x_2, \cdots x_n).$$

Определение 7. Функция $f(x_1, x_2, \cdots x_{i-1}, x_i, x_{i+1}, \cdots x_n)$ существенно зависит от аргумента x_i, если имеет место соотношение:

$$f(x_1, x_2, \cdots x_{i-1}, 0, x_{i+1}, \cdots x_n) \neq f(x_1, x_2, \cdots x_{i-1}, 1, x_{i+1}, \cdots x_n),$$

В противном случае говорят, что от x_i функция зависит несущественно и x_i является ***фиктивным аргументом***. ФАЛ не изменится, если к ее аргументам дописать любое число фиктивных аргументов, или зачеркнуть все фиктивные аргументы.

Теорема 7.1 Число различных ФАЛ, зависящих от n аргументов конечно и равно 2^{2^n}.

Для доказательства составим таблицу:

x_1	x_2	…	x_{n-1}	x_n	$f(x_1, x_2, \cdots x_n)$
0	0	0	0	0	α_1
0	0	0	0	1	α_2
0	0	0	1	0	α_3
0	0	0	1	1	α_4

…	…	…	…	…	α_k
1	1	1	1	0	α_{2^n-1}
1	1	1	1	1	α_{2^n}

В этой таблице справа стоит одна из ФАЛ, зависящая от n аргументов. Так как α_i принимает значение либо 0, либо 1, то задавая тот или иной конкретный двоичный набор $\langle\alpha_1, \alpha_2, \cdots \alpha_{2^n}\rangle$, мы будем тем самым задавать одну из возможных функций алгебры логики. Но тогда различное число таких наборов равно $2^{(2^n)}$. Теорема доказана.

Значения функции могут быть заданы не на всех возможных наборах аргументов. На некоторых наборах значения ФАЛ могут быть не определены. Такие ФАЛ называют ***не полностью определенными*** или ***недоопределенными.*** Функцию можно доопределить. Если ФАЛ не определена на m наборах аргументов, то путем ее произвольного доопределения можно получить 2^m различных полностью определенных функций.

7.2 Элементарные функции алгебры логики

В число ФАЛ, подсчитываемых с помощью теоремы 7.1 входят как функции существенно зависящие от всех n аргументов, так и функции, для которых часть из этих аргументов фиктивна. Например, для n=1 имеем $2^{2^1} = 4$ различных функций, приведенных в следующей таблице:

x_1	$f_1(x_1)$	$f_2(x_1)$	$f_3(x_1)$	$f_4(x_1)$
0	0	1	0	1
1	0	1	1	0

Из таблицы видно, что только $f_3(x_1)$ и $f_4(x_1)$ существенно зависят от x_1, а для $f_1(x_1)$ и $f_2(x_1)$ этот единственный аргумент является фиктивным. Функция $f_3(x_1)$, повторяющая значение логической переменной - ***тождественная функция*** ($f_3(x_1) = x_1$), а функция $f_4(x_1)$- принимающая значения, обратные значениям x_1 - ***логическое отрицание,*** или ***функция отрицания***, или ***инверсии,*** или функция ***НЕ*** ($f_4(x_1) = \bar{x}_1$).

Теорема 7.2. Число ФАЛ, существенно зависящих от n аргументов, определяется следующим рекуррентным соотношением:

$$A_n = 2^{2^n} - C_n^{n-1} \cdot A_{n-1} - C_n^{n-2} \cdot A_{n-2} - \ \ldots - C_n^1 \cdot A_1 - A_0, \qquad (7.2)$$

где $C_n^k = \frac{n!}{k!(n-k)!}$= - биномиальные коэффициенты, A_i- число ФАЛ, существенно зависящих от i аргументов.

Правая часть соотношения есть разность между числом всех ФАЛ и суммой всех ФАЛ, существенно завис4ящих от любого числа аргументов, меньших чем n.

Пример Найти число ФАЛ, существенно зависящих от 3-х переменных.

Имеем: A_0=2.

Действительно: $A_0 = 2^{2^0} = 2^1 = 2.$

$$A_1 = 2^{2^1} - A_0 = 4 - 2 = 2.$$

$$A_2 = 2^{2^2} - C_2^1 A_1 - A_0 = 2^4 - \frac{2!}{1!1!}2 - 2 = 10.$$

$$A_3 = 2^{2^3} - C_3^2 A_2 - C_3^1 A_1 - A_0 = 2^8 - 3 \cdot 10 - 3 \cdot 2 - 2 =$$
$$= 256 - 30 - 6 - 2 = 218.$$

Итак, в случае n=0 имеем в силу теоремы 1.1 две функции, совпадающие с константами 0 и 1.

$$f_1 = 0; \qquad f_2 = 1 \qquad (7.3)$$

При n =1 имеем 2 функции, существенно зависящих от x

$$f_3(x) = x; \qquad f_4(x) = \bar{x} \text{ (не ' } x \text{ ')}. \qquad (7.4)$$

Рассмотрим логические функции от 2-х переменных (Таблица 7.1):

Таблица 7.1

Функции	x_1x_2	x_1x_2	x_1x_2	x_1x_2	Примечание.
	0 0	0 1	1 0	1 1	
f_1	0	0	0	0	$f \equiv 0$
f_2	0	0	0	1	$f = x_1 x_2 = x_1 \wedge x_2 = x_1 \& x_2$ конъюнкция.
f_3	0	0	1	0	$f = x_1 \bar{x}_2$ (запрет x_2)
f_4	0	0	1	1	$f = x_1$ (тождественность x_1)
f_5	0	1	0	0	$f = \bar{x}_1 x_2$ (запрет x_1)
f_6	0	1	0	1	$f = x_2$ (тождественность x_2)
f_7	0	1	1	0	$f = (x_1 \oplus x_2)$ (сложение по модулю 2)
f_8	0	1	1	1	$f = x_1 \vee x_2$ (дизъюнкция)
f_9	1	0	0	0	$f = x_1 \downarrow x_2$ (функция Пирса)
f_{10}	1	0	0	1	$f = (x_1 \equiv x_2)$ (равнозначность)
f_{11}	1	0	1	0	$f = \bar{x}_2$ (отрицание x_2)
f_{12}	1	0	1	1	$f = (x_2 \to x_1)$ (импликация)
f_{13}	1	1	0	0	$f = \bar{x}_1$ (отрицание x_1)
f_{14}	1	1	0	1	$f = (x_1 \to x_2)$ (импликация)
f_{15}	1	1	1	0	$f = (x_1 / x_2)$ (функция Шеффера)

f_{16}	1	1	1	1	$f \equiv 1$

В силу теоремы 7.2 имеем 10 различных функций, существенно зависящих от аргументов x_1 и x_2. Из них рассмотрим семь, которые играют большую роль в построении теории ФАЛ и ее приложениях. (Помимо рассмотренных 6.3 и 6.4).

а). Функция $f = x_1 x_2 = x_1 \wedge x_2 = x_1 \& x_2$ ***конъюнкция*** **(*логическое умножение,*** или ***функция И, AND***) истинна тогда и только тогда, когда и x_1 и x_2 истинны.

x_1	x_2	f
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

б). Функция $f = x_1 \vee x_2$ - ***дизъюнкция*** **(*логическое сложение,*** или ***функция ИЛИ, OR*)** истинна тогда, и только тогда, когда истинны или x_1, или x_2, или обе переменные.

x_1	x_2	f
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

в). Функция $f = (x_1 \oplus x_2)$ ***функция сложения по модулю*** **2 (**или ***функция разноименности,*** или ***функция исключающее ИЛИ, XOR*)** истинна тогда, когда истинна или x_1, или x_2, но не обе вместе

x_1	x_2	f
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

г).Функция - $f = (x_1 \equiv x_2) = (x_1 \sim x_2)$ ***функция тождественности*** которая истинна тогда и только тогда, когда обе переменные или истинны, или ложны.

x_1	x_2	f
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**1**

д.) Функция $f = (x_2 \to x_1)$ **(*импликация* x_2 *в* x_1),** которая ложна тогда и только тогда, когда x_2 истинна, а x_1 ложна.

x_1	x_2	$f = x_1 \to x_2$	$f = x_2 \to x_1$
0	0	**1**	**1**
0	1	**1**	**0**
1	0	**0**	**1**
1	1	**1**	**1**

е.) Функция $f = x_1 \downarrow x_2$ - ***функция Пирса* (*Вебба*),** (или функция ***ИЛИ-НЕ***, ***NOR***) которая истинна тогда и только тогда, когда x_1 и x_2 ложны.

x_1	x_2	f
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

ж.) Функция $f = (x_1 / x_2)$ ***функция Шеффера,*** **(**или функция ***И-НЕ, NAND***) которая ложна только тогда, когда x_1 и x_2 истинны.

x_1	x_2	f
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

Для наглядности рассмотренные функции сведены в общую таблицу:

x_1	x_2	$x_1 \& x_2$	$x_1 \vee x_2$	$x_1 \oplus x_2$	$x_1 \downarrow x_2$	$x_1 \to x_2$	$x_2 \to x_1$	$x_1 \equiv x_2$	x_1 / x_2
0	0	0	0	0	1	1	1	1	1
0	1	0	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	0	1
1	1	1	1	0	0	1	1	1	0

Рассмотренные функции алгебры логики называются ***элементарными*** и позволяют строить новые ФАЛ.

7.3 Выражение одних элементарных функций через другие.

Очень часто для преобразования выражений алгебры логики используются выражение одних элементарных функций через другие. Для доказательства равносильности формул воспользуемся методом

непосредственной проверки совпадения функций, образующих правую и левую часть доказываемых соотношений (См. определение 6)

1. $x_1 \to x_2 = \bar{x}_1 \vee x_2$ (7.5)

x_1	x_2	$x_1 \to x_2$
0	0	1
0	1	1
1	0	0
1	1	1

x_1	x_2	$\bar{x}_1$	$\overline{x_1} \vee x_2$
0	0	1	1
0	1	1	1
1	0	0	0
1	1	0	1

2. $x_1 \oplus x_2 = \overline{x_1 \equiv x_2}$ (7.6)

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

x_1	x_2	$x_1 \sim x_2$	$x_1 \sim x_2$
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	0

3. $x_1 \equiv x_2 = (\bar{x}_1 \vee x_2)(x_1 \vee \bar{x}_2)$ (7.7)

x_1	x_2	$x_1 \sim x_2$
0	0	1
0	1	0
1	0	0
1	1	1

$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_1 \vee x_2$	$x_1 \vee \bar{x}_2$	$(\bar{x}_1 \vee x_2)(x_1 \vee \bar{x}_2)$
1	1	1	1	1
1	0	1	0	0
0	1	0	1	0
0	0	1	1	1

4. $x_1 = \bar{\bar{x}}_1$ (7.8)

x_1	$\bar{x}_1$	$\bar{\bar{x}}_1$
0	1	0
1	0	1

5. $x_1 \oplus x_2 = \overline{(\bar{x}_1 \vee x_2)(x_1 \vee \bar{x}_2)}$ (7.9)

6. $x_1 x_2 = \overline{\bar{x}_1 \vee \bar{x}_2}$ (7.10)

x_1	x_2	$x_1 x_2$
0	0	0
0	1	0

x_1	x_2	$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_1 \vee \bar{x}_2$	$\overline{\bar{x}_1 \vee \bar{x}_2}$
0	0	1	1	1	0
0	1	1	0	1	0

1	0	0		1	0	0	1	1	0
1	1	1		1	1	0	0	0	1

7. $x_1 \vee x_2 = \overline{\bar{x}_1 \bar{x}_2}$ (7.11)

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_1\bar{x}_2$	$\overline{\bar{x}_1\bar{x}_2}$
0	0	1	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	0	1

Выражения (7.10) и (7.11) известны как ***правила де Моргана***.

Аналогично можно установить ряд взаимоотношений между другими функциями алгебры логики.

Теперь, используя установленные “взаимоотношения” между элементарными ФАЛ и аксиомы алгебры логики, можно аналитически выразить одни ФАЛ через другие. Например, используя “табличный метод” доказательства, было установлено, что $x_1 \oplus x_2 = \overline{x_1 \equiv x_2}$. Тогда справедливо отношение: $\overline{x_1 \oplus x_2} = \overline{\overline{x_1 \equiv x_2}}$. Учитывая, что $\bar{\bar{x}} = x$, получаем следующее выражение: $\overline{x_1 \oplus x_2} = x_1 \equiv x_2$.

Аналогично, используя правила де Моргана получаем следующее выражение для функции исключающее ИЛИ:

$$x_1 \oplus x_2 = \overline{(\bar{x}_1 \vee x_2)(x_1 \vee \bar{x}_2)} = \overline{(\bar{x}_1 \vee x_2)} \vee \overline{(x_1 \vee \bar{x}_2)} = x_1\bar{x}_2 \vee \bar{x}_1 x_2.$$

7.4 Законы алгебры логики

Законы алгебры логики записываются в виде формул, которые позволяют производить эквивалентные преобразования логических выражений. Используя основные положения АЛ, нетрудно убедиться в справедливости следующих основных восьми правил, которые часто называют основными законами алгебры логики.

Закон тождества.

$x=x$.

Законы идемпотентности.

$x \vee x = x$;

$x \cdot x = x$.

Закон исключения констант

$x \vee 1 = 1$;

$x \vee 0 = x$;

$x \cdot 1 = x$;

$x \cdot 0 = 0$.

Закон поглощения

$x \vee x \cdot y = x;$

$x \cdot (x \vee y) = x.$

Закон противоречия

$x \cdot \bar{x} = 0.$

Закон исключенного третьего

$x \vee \bar{x} = 1.$

Закон двойного отрицания

$\bar{\bar{x}} = x.$

Законы Де-Моргана:

$x \vee y = \overline{\bar{x} \cdot \bar{y}};$

$x \cdot y = \overline{\bar{x} \vee \bar{y}}.$

7.5 Свойства конъюнкции, дизъюнкции.

Функции конъюнкции (функция ***И***), дизъюнкции (функция ***ИЛИ***) обладают рядом свойств, аналогичным свойствам обычных операций умножения и сложения. Для них справедливы:

1. ***Свойство ассоциативности*** (сочетательный закон):

$$x_1 \cdot (x_2 \cdot x_3) = (x_1 \cdot x_2) \cdot x_3 = (x_1 \cdot x_3) \cdot x_2 \cdot$$

$$x_1 \vee (x_2 \vee x_3) = (x_1 \vee x_2) \vee x_3 = (x_1 \vee x_3) \vee x_2 \cdot$$

2. ***Свойство коммутативности*** (переместительный закон):

$$x_1 \cdot x_2 = x_2 \cdot x_1 \cdot$$

$$x_1 \vee x_2 = x_2 \vee x_1 \cdot$$

3. ***Свойство дистрибутивности*** (распределительный закон):

- для конъюнкции относительно дизъюнкции:

$$x_1 \,\&\, (x_2 \vee x_3) = (x_1 \,\&\, x_2) \vee (x_1 \,\&\, x_3) \cdot$$

- для дизъюнкции относительно конъюнкции:

$$x_1 \vee x_2 \,\&\, x_3 = (x_1 \vee x_2) \,\&\, (x_1 \vee x_3) \cdot$$

Действительно: $(x_1 \vee x_2) \cdot (x_1 \vee x_3) = x_1 \cdot x_1 \vee x_1 \cdot x_3 \vee x_2 \cdot x_1 \vee x_2 \cdot x_3 =$

$x_1 \vee x_1 \cdot x_3 \vee x_1 \cdot x_2 \vee x_2 \cdot x_3 = x_1(1 \vee x_3 \vee x_2) \vee x_2 \cdot x_3 = x_1 \vee x_2 \cdot x_3 \cdot$

Обобщение формул (7.10) и (7.11) позволяет получить формулы, известные как ***законы Де-Моргана***:

$$x_1 \vee x_2 \vee x_3 \vee \cdots \vee x_n = \overline{\bar{x}_1 \,\&\, \bar{x}_2 \,\&\, \bar{x}_3 \,\&\, \cdots \,\&\, \bar{x}_n}. \qquad (7.12)$$

$$x_1 \,\&\, x_2 \,\&\, x_3 \,\&\, \cdots \,\&\, x_n = \overline{\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \cdots \vee \bar{x}_n}. \qquad (7.13)$$

Именно из логических функций устанавливаются соотношения, известные как ***законы (правила) поглощения***, уже приведенные ранее.

$$x_1 \vee x_1 \cdot x_2 = x_1;$$

$$x_1(x_1 \vee x_2) = x_1 .$$

Доказательство этих соотношений не вызывает затруднений. Например, для первого соотношения имеем:

$$x_1 \vee x_1 \cdot x_2 = x_1 \cdot (1 \vee x_2) = x_1 \cdot 1 = x_1 .$$

Второе соотношение доказывается аналогично:

$$x_1(x_1 \vee x_2) = x_1 \cdot x_1 \vee x_1 \cdot x_2 = x_1 \vee x_1 \cdot x_2 = x_1 \cdot (1 \vee x_2) = x_1 .$$

Из логических функций устанавливается ***правило склеивания***:

$$x_1 x_2 \vee x_1 \bar{x}_2 = x_1$$

Также для логических функций установлено ***правило вычеркивания***:

$$x_1 \vee \bar{x}_1 \cdot x_2 = x_1 \vee x_2 .$$

Действительно:

$$x_1 \vee \bar{x}_1 \cdot x_2 = x_1 \cdot 1 \vee \bar{x}_1 x_2 = x_1 \cdot (x_2 \vee \bar{x}_2) \vee \bar{x}_1 x_2 = x_1 \cdot x_2 \vee x_1 \bar{x}_2 \vee \bar{x}_1 x_2 =$$
$$= (x_1 x_2 \vee x_1 \bar{x}_2) \vee (x_1 x_2 \vee \bar{x}_1 x_2) = x_1(x_2 \vee \bar{x}_2) \vee x_2(x_1 \vee \bar{x}_1) =$$
$$= x_1 \cdot 1 \vee x_2 \cdot 1 = x_1 \vee x_2 .$$

Знание свойств, законов и правил элементарных ФАЛ необходимо для аналитического описания функций алгебры логики, их преобразований.

В алгебре логики часто используется выражение ***терм***, который можно определить следующим образом:

1. Логические переменные или их отрицания, связанные знаком конъюнкции или дизъюнкции, называются ***термами***.
2. Число переменных, входящих в рассматриваемый терм, называется ***рангом терма***.

Например, $x_4 x_2 \bar{x}_1 x_0$ – конъюнктивный терм 4-го ранга, а $\bar{x}_4 \vee x_3 \vee x_1$ – дизъюнктивный терм 3-го ранга. Выражение $x_3 x_2 \bar{x}_0 \vee x_1$ – термом не является и представляет собой два терма: $x_3 x_2 \bar{x}_0$ и x_1.

7.6 Представление функций алгебры логики

Существует несколько способов заданий функций алгебры логики. Она может быть задана словесно, алгебраическим выражением, и таблицей, которая называется таблицей истинности (***truth table***). Наряду с табличным методом в инженерной практике наиболее широко используются аналитические методы представления ФАЛ. Существуют теоремы о представлении функции алгебры в тех или иных формах. Наиболее известны дизъюнктивное и конъюнктивное представления ФАЛ, согласно которым доказано, что любая функция алгебры логики может быть представлена:

- ***в дизъюнктивной нормальной форме*** (***ДНФ***);
- ***в конъюнктивной нормальной форме*** (***КНФ***).

В основе представления ФАЛ в ДНФ или КНФ лежат теоремы о представлении любой ФАЛ в ***совершенной нормальной дизъюнктивной форме*** (***СНДФ***) и в ***совершенной нормальной конъюнктивной форме*** (***СНКФ***).

Пусть ФАЛ $f(x_{n-1}, x_{n-2}, \cdots, x_1, x_0)$ задана в табличной форме. Алгоритм перехода от таблично заданной ФАЛ к СНДФ Алгоритм перехода от таблично заданной ФАЛ к СНДФ заключается в следующем:

1. Выбрать в таблице задания функции все наборы аргументов, на которых функция обращается в единицу.
2. Записать конъюнкцию (минтерм) всех переменных, соответствующего набора. Если при этом аргумент x_i на данном наборе равен 1, он в конъюнкцию записывается без изменения. Если же x_i=0, то в конъюнкцию вписывается его отрицание.
3. Все полученные конъюнкции соединяются между собой знаками дизъюнкции.

Пример. Пусть ФАЛ $f(a, b, c)$ задана таблицей истинности:

Таблица 7.2

Номер набора	a	b	c	$f(a,b,c)$	минтерм	макстерм
0	0	0	0	**1**	$\bar{a}\bar{b}\bar{c}$	
1	0	0	1	**1**	$\bar{a}\bar{b}c$	
2	0	1	0	**0**		$(a \vee \bar{b} \vee c)$
3	0	1	1	**1**	$\bar{a}bc$	
4	1	0	0	**0**		$(\bar{a} \vee b \vee c)$
5	1	0	1	**0**		$(\bar{a} \vee b \vee \bar{c})$
6	1	1	0	**0**		$(\bar{a} \vee \bar{b} \vee c)$
7	1	1	1	**1**	abc	

Как видно из таблицы, на четырех наборах (0, 1, 3 и 7) функция равна единице, и, следовательно, согласно приведенному алгоритму, получаем:

$$f(a, b, c) = \bar{a}\bar{b}\bar{c} \vee \bar{a}\bar{b}c \vee \bar{a}bc \vee abc.$$

Полученная СНДФ легко приводится и к ДНФ путем аналитических преобразований с использованием основных свойств и законов алгебры логики:

$$f(a, b, c) = \bar{a}\bar{b}\bar{c} \vee \bar{a}\bar{b}c \vee \bar{a}bc \vee abc = \left(\bar{a}\bar{b}\bar{c} \vee \bar{a}\bar{b}c\right) \vee (\bar{a}bc \vee abc) =$$

$$\bar{a}\bar{b}(\bar{c} \vee c) \vee bc(\bar{a} \vee a) = \bar{a}\bar{b} \cdot 1 \vee bc \cdot 1 = \bar{a}\bar{b} \vee bc.$$

Т.е. получена равносильная СНДФ функция, содержащая всего уже два терма, каждый из которых 2-го ранга.

Алгоритм построения СНКФ:

1. Выбрать в таблице задания функции все наборы аргументов, на которых функция обращается в нуль.
2. Записать дизъюнкцию соответствующего набора переменных. Если при этом логическая переменная x_i на данном наборе равна 0, то она вписывается в дизъюнкцию без изменения. Если же x_i на рассматриваемом наборе равна 1, то в с дизъюнкцию выписывается ее отрицание.
3. Все полученные дизъюнкции соединяются между собой знаками конъюнкций.

Для функции, заданной таблицей 7.1, на четырех наборах (2, 4, 5, 6) функция равна 0 и, следовательно, СНКФ будет равна конъюнкции 4-х дизъюнктивных термов:

$$f(a, b, c = (a \vee \bar{b} \vee c) \cdot (\bar{a} \vee b \vee c) \cdot (\bar{a} \vee b \vee \bar{c}) \cdot (\bar{a} \vee \bar{b} \vee c).$$

А КНФ данная функция будет иметь вид:

$$f(a, b, c) = (\bar{a} \vee b) \cdot (\bar{b} \vee c).$$

7.7 Логические элементы и их применение

Одним из самых широких приложений алгебры логики является разработка дискретных, цифровых устройств различного назначения и, в частности, средств вычислительной техники.

Техническим аналогом элементарных логических функций являются ***логические элементы*** (***ЛЭ***), представляющие собой физические устройства, выполняющие логические операции над сигналами, представленными в физической форме. Т.е. в этих устройствах логические единица и нуль представлены физическими сигналами различной величины, одна из величин соответствует логической единице, другая – логическому нулю. Например, различные величины давления: p_1 и p_2 (пневматические или гидравлические логические элементы); различные величины интенсивности освещения: s_1, s_2 (оптические или оптоэлектронные логические элементы). Но доминирующее значение в технике имеют ***электронные логические элементы***, которые представляют собой определенные электронные схемы, реализующие логические операции над логическими переменными, представленными в этом случае в виде электрических сигналов. Имеет место несколько

способов представления логических единицы и нуля в виде электрических сигналов, но преимущественно используется потенциальный способ представления логических переменных, при котором значениям логической единицы и нуля соответствуют вполне определенные уровни напряжения: U^1 и U^0. Потенциальные логические элементы (схемы) в настоящее время составляют основу системы изделий микроэлектроники для цифровой, вычислительной техники. Они представлены в виде серии микросхем различных степеней интеграции, предназначенных для построения цифровых устройств различного назначения, и в частности – ЭВМ различных классов.

По виду реализуемой логической функции основные логические элементы условно разделяют на элементы одноступенчатой логики, реализующие функции ***И*** (конъюнкция), ***ИЛИ*** (дизъюнкция), ***НЕ*** (отрицание), ***И-НЕ***, ***ИЛИ-НЕ***, и на элементы двухступенчатой логики, реализующие функции ***И-ИЛИ***, ***ИЛИ-И***, ***И-ИЛИ-НЕ***, ***ИЛИ-И-НЕ***.

Число входов ***ЛЭ*** соответствует числу входных переменных, воспроизводимой им логической функции. Промышленно выпускаемые логические элементы имеют ограниченное количество входов (коэффициент разветвления по входу в большинстве случаев равен 2 - 8) и ограниченную нагрузочную способность. Но имеются известные схемотехнические решения увеличения коэффициента разветвления по входу. Практическая реализация логических схем сегодня осуществляется, как правило, с применением интегральной схемотехники, среди которой наиболее широкое распространение получили элементы потенциального типа, логические переменные 0 и 1 в которых, как уже отмечалось, отображаются двумя уровнями напряжения: U^0 и U^1 соответственно. Для логического описания цепей не требуются абсолютные значения U^0 и U^1. Необходимо лишь условиться, какой из них отображает логический нуль и какой - логическую единицу.

Возможно два варианта представления логических переменных с помощью двух уровней напряжения. В первом варианте за 1 принимается высокий уровень напряжения, обозначаемый латинской буквой ***H*** (от англ. high). Во втором варианте логическая единица отображается низким уровнем напряжения - ***L*** (от англ. low). Первый вариант называют соглашением положительной логики ($U^0 < U^1$), второй - соглашением отрицательной логики ($U^0 > U^1$).

Изменение способа представления переменных ведет к изменению логической функции, выполняемой данным элементом (принцип

двойственности). Например, пусть имеется элемент, функционирующий следующим образом:

x_1	x_2	y
L	L	**L**
L	H	**L**
H	L	**L**
H	H	**H**

где x_1 и x_2 - входные сигналы, y - выходной сигнал

Приняв положение положительной и отрицательной логики, соответственно получаем:

При соглашении положительной логики *L*=”0”, *H*=”1” и тогда таблица будет иметь вид:

x_1	x_2	y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Согласно определению, видно, что в данном случае элемент выполняет операцию ***И.***

При соглашении отрицательной логики ***L***=”1”, ***H***=”0” и в этом случае получаем:

x_1	x_2	y
1	1	**1**
1	0	**1**
0	1	**1**
0	0	**0**

В этом случае элемент, как видно, выполняет операцию ***ИЛИ***.

Т.е. при соглашении положительной логики получаем $y = x_1 \cdot x_2$. Действительно, инвертируя все переменные выражения $y = x_1 \cdot x_2$, получаем, $\bar{y} = \bar{x}_1 \cdot \bar{x}_2$. Тогда $\bar{\bar{y}} = \overline{\bar{x}_1 \cdot \bar{x}_2}$. Откуда по законам алгебры логики получаем $y = \overline{\bar{x}_1 \cdot \bar{x}_2} = \bar{\bar{x}}_1 \vee \bar{\bar{x}}_2 = x_1 \vee x_2$. Т.е. для отрицательной логики этот элемент выполняет операцию ***ИЛИ***.

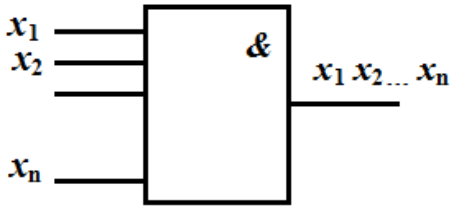
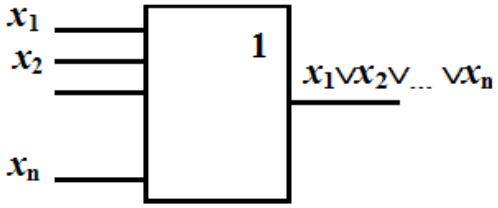
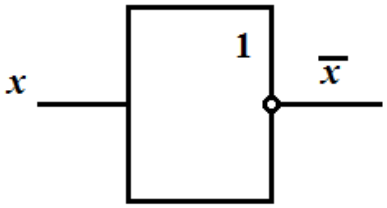
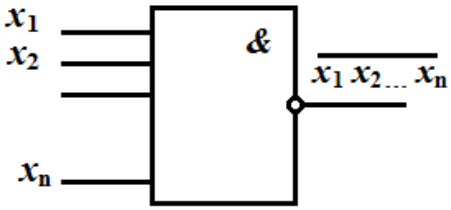
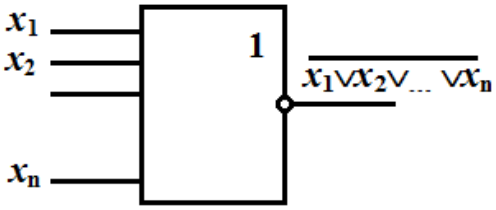
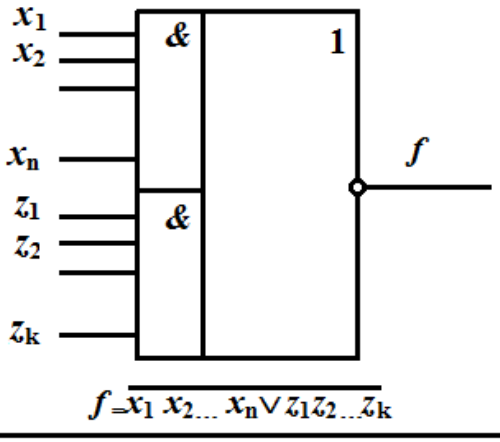
Аналогично можно установить соответствие между выполняемыми логическими элементами функциями при соглашении положительной и отрицательной логики для всех элементарных ФАЛ.

Принятые условные обозначения некоторых основных ЛЭ приведены в таблице 7.3.

7.8 Функционально полные системы функций алгебры логики

Система элементарных ФАЛ $\varphi_1, \varphi_2, \cdots \varphi_m$ называется функционально полной, или ***базисом***, если любую функцию алгебры логики можно представить в виде их ***суперпозиции функций***. Имея логические элементы, осуществляющие операции $f_1 \cdots f_{16}$ (см. табл. 7.1) можно выполнить любую сложную функцию. Однако условие наличия 16 различных типов логических элементов, каждый из которых реализует одну из 16 элементарных операций, является условием, достаточным, но необходимым

Таблица 7.3

Тип логического элемента	Обозначение	Выполняемая функция
И		Конъюнкция
ИЛИ		Дизъюнкция
НЕ		Инверсия
И-НЕ		Отрицание конъюнкции
ИЛИ-НЕ		Отрицание дизъюнкции
И-ИЛИ-НЕ		Отрицание дизъюнкции конъюнкций

Для синтеза логического устройства любой сложности можно ограничиваться меньшим набором элементарных функций. Последовательно исключая из базиса избыточные функции, можно получить ***минимальный базис***. ***Под минимальным базисом*** понимают такой набор функций, исключение из которого любой функций превращает этот набор в неполную систему функций.

Возможны различные базисы и минимальные базисы, различающиеся числом входящих в них функций и видом этих функций. Выбор базиса связан с тем, насколько просто, удобно и экономично технически выполнять элементы, реализующие элементарные функции, которые входят в выбранный базис, и устройство в целом.

Функционально полными системами являются базисы:

Базис	Состав
1	***И, ИЛИ, НЕ***
2	***И, НЕ***
3	***ИЛИ, НЕ***
4	***И-НЕ***
5	***ИЛИ-НЕ***
6	***И-ИЛИ-НЕ***

Базис ***И***, ***ИЛИ***, ***НЕ*** принято называть ***основным***, так как любая сложная переключательная функция может быть записана в СДНФ или СКНФ. Базисы ***И, НЕ*** и ***ИЛИ, НЕ*** называют ***универсальными***. Эти базисы приобрели большое значение в связи с широким использованием интегральных логических элементов при построении логических устройств.

Для набора логических элементов можно ввести понятие ***функциональной полноты***, подобно тому, как это было сделано для случая ФАЛ. Набор ***ЛЭ*** обладает функциональной полнотой, если с помощью этого набора можно реализовать схему с любым законом функционирования.

Т.е. любая комбинационная схема может быть построена с применением логических элементов типа ***И***, ***ИЛИ***, ***НЕ***, совокупность которых является функционально полной системой (т.е. базисом), а также только и из элементов типа ***И-НЕ*** (функция $f(x_1, x_2, \cdots x_n) = \overline{x_1 \cdot x_2 \cdot \cdots \cdot x_n} = \overline{\bigwedge_{i=1}^{n} x_i}$), или из элементов типа ***ИЛИ-НЕ*** (функция $f(x_1, x_2, \cdots x_n) = \overline{x_1 \vee x_2 \vee \cdots \vee x_n} = \overline{\bigvee_{i=1}^{n} x_i}$), или из элементов

типа ***И-ИЛИ-НЕ*** (функция $f(x_1, x_2, \cdots x_n) = \overline{\bigvee_{j=1}^{k} \cdot \bigwedge_{i=1}^{n} x_{ij}}$), каждая из которых представляет функционально полною систему.

В цифровой схемотехнике наиболее активно используются логические элементы и устройства реализуемые в базисе ***И-НЕ*** или в базисе ***ИЛИ-НЕ***, которые представляют, как уже отмечалось, функционально полный базис, т.е. позволяют реализовать любую логическую функцию. Например, на рис.7.1 приведена реализация элементарных ФАЛ ***НЕ***, ***ИЛИ И*** на базе логического элемента **2*И-НЕ***.

Действительно, в базисе **2*И-НЕ*** имеем:

а) $\bar{x} = \overline{x \cdot x}$, т.е. инвертор реализуется элементом **2*И-НЕ*** в соответствии с рис.6.1 а);

б) $x_1 \vee x_2 = \overline{\bar{x}_1 \cdot \bar{x}_2} = \overline{\overline{x_1 \cdot x_1} \cdot \overline{x_2 \cdot x_2}}$, т.е. дизъюнкция реализуется тремя элементами **2*И-НЕ*** (рис.6.1 б);

в) $x_1 \cdot x_2 = \overline{\overline{x_1 \cdot x_2}} = (x_1 / x_2)/(x_1 / x_2)$, т.е. конъюнкция реализуется двумя элементами **2*И-НЕ*** (рис.6.1 в).

С

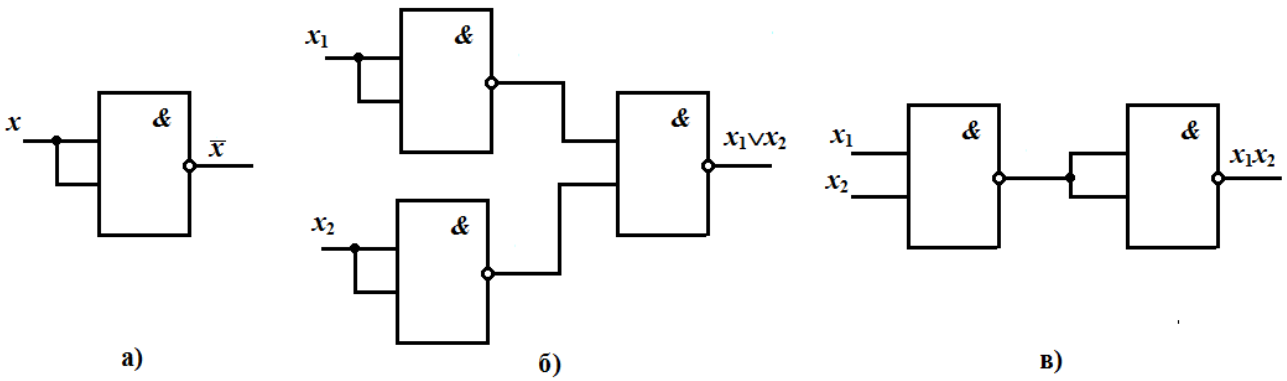


Рис. 7.1 Реализация элементов основного базиса на элементах **2*И-НЕ***
а) инвертор (***НЕ***); б) дизъюнкция (***ИЛИ***); в) конъюнкция (***И***).

На рис. 7.2 приведена реализация элементарных ФАЛ ***НЕ***, ***И ИЛИ*** на базе логического элемента **2*ИЛИ-НЕ***.

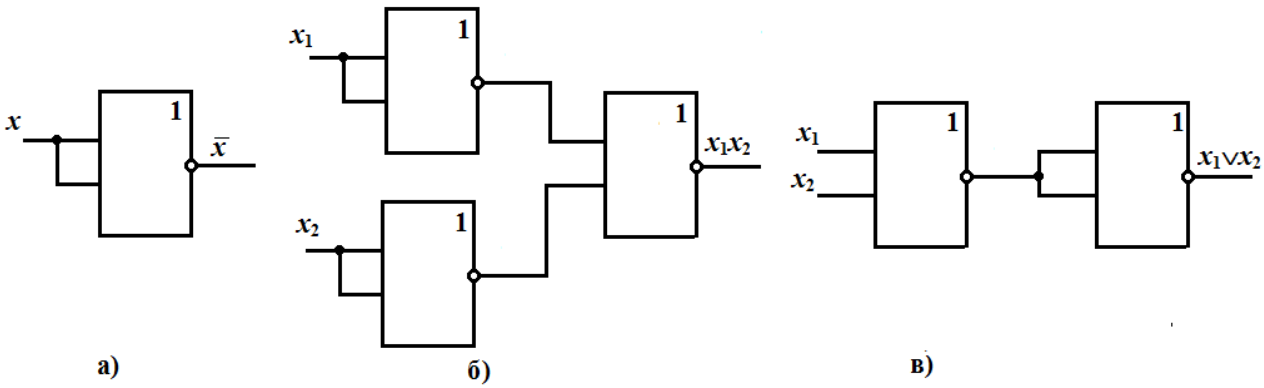


Рис. 7.2 Реализация элементов основного базиса на элементах **2*ИЛИ-НЕ***
а) инвертор (***НЕ***); б) конъюнкции (***И***); в) дизъюнкции (***ИЛИ***).

В базисе **2*ИЛИ-НЕ*** получаем:

а) $\bar{x} = \overline{x \vee x}$, т.е. инвертор реализуется элементом **2*ИЛИ-НЕ*** в соответствии с рис.6.2 а);

б) $x_1 \cdot x_2 = \overline{\overline{x_1 \vee x_1} \vee \overline{x_2 \vee x_2}} = \overline{\bar{x}_1 \vee \bar{x}_2}$, т.е. конъюнкция реализуется тремя элементами **2*ИЛИ-НЕ*** (рис. 7.2 б);

в) $x_1 \vee x_2 = \overline{\overline{x_1 \vee x_2}}$ т.е. дизъюнкция реализуется двумя элементами **2*ИЛИ-НЕ*** (рис. 7.2 в);

Рассмотрим пример схемной реализации логической функции, заданной, например, табличным методом (таблица 7.4).

В СНДФ функция будет иметь вид:

$$y = \bar{x}_1\bar{x}_2x_3\bar{x}_4 \vee \bar{x}_1\bar{x}_2x_3x_4 \vee \bar{x}_1x_2\bar{x}_3\bar{x}_4 \vee x_1\bar{x}_2\bar{x}_3x_4 \vee x_1\bar{x}_2x_3\bar{x}_4 \vee x_1\bar{x}_2x_3x_4 \vee$$
$$\vee x_1x_2\bar{x}_3\bar{x}_4 \vee x_1x_2x_3x_4.$$

Таблица 7.4

x_1	x_2	x_3	x_4	y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1

1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Минимизируя данную функцию с помощью карты Карно (рис. 7.5), получаем: $y = \bar{x}_2 x_3 \vee x_1 x_3 x_4 \vee x_1 \bar{x}_2 x_4 \vee x_2 \bar{x}_3 \bar{x}_4$.

Реализуем данную функцию в базисе элементов ***И-НЕ***:

$$y = \overline{\overline{\bar{x}_2 x_3} \cdot \overline{x_1 x_3 x_4} \cdot \overline{x_1 \bar{x}_2 x_4} \cdot \overline{x_2 \bar{x}_3 \bar{x}_4}}$$

Из последнего выражения видно, что для реализации данной функции необходимо иметь три элемента типа ***3И-НЕ***, один элемент типа ***2И-НЕ***, один элемент типа ***4И-НЕ*** и три элемента типа ***НЕ*** (инвертора).

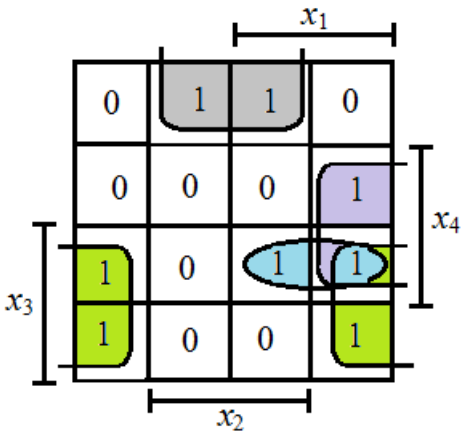


Рис.7.5

На рисунке 7.6 приведена соответствующая логическая схема, реализующая заданную функцию в базисе элементов ***И-НЕ***.

Аналогично можно данную функцию реализовать и в других элементных базисах.

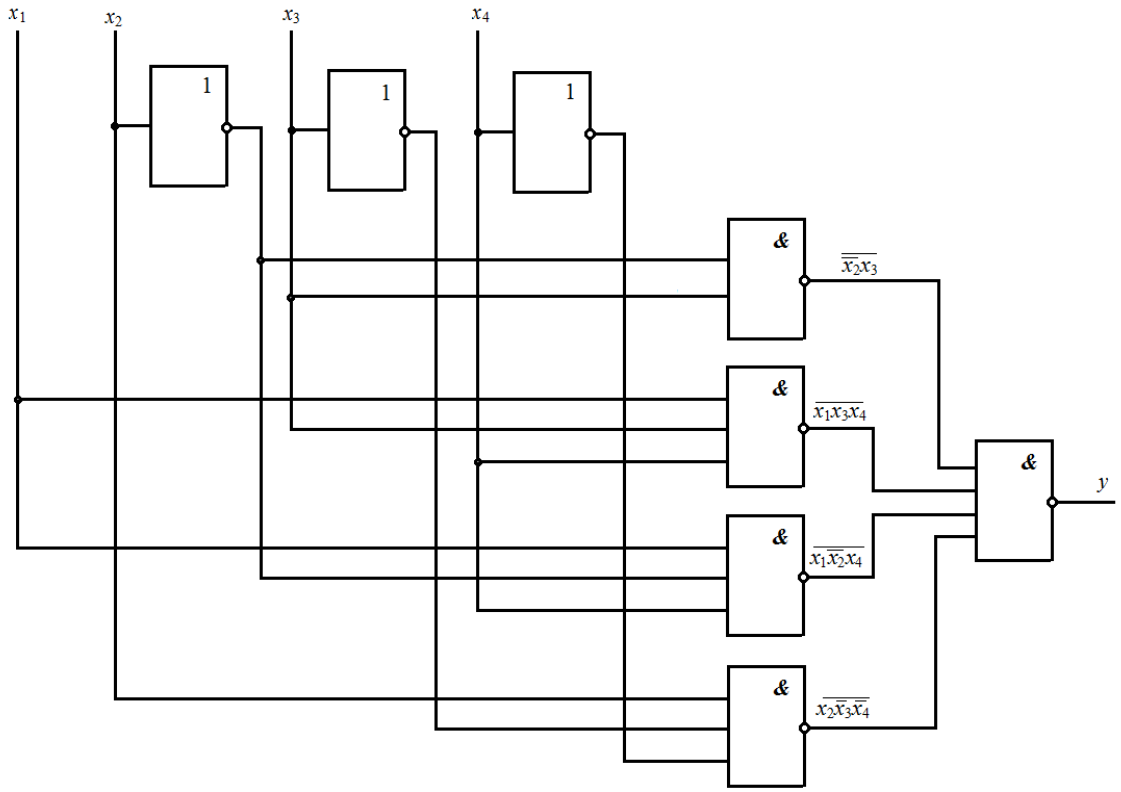

x_1
x_2
x_3
x_4
1
1
1
&
&
&
&
&
$\overline{\overline{x_2}x_3}$
$\overline{x_1x_3x_4}$
$\overline{x_1\overline{x_2}x_4}$
$\overline{x_2\overline{x_3}\overline{x_4}}$
y

Рис. 7.6

Лекция 8

Процессор. Архитектура ЭВМ

Процессор - это устройство, выполняющее последовательность команд (программу) и координирующее действия других устройств.

В ЭВМ может быть несколько процессоров. Другие устройства ЭВМ, например, ОЗУ, также могут иметь свои процессоры. Процессор, управляющий всей ЭВМ, называется **центральным**. Процессоры, размещённые на одной интегральной схеме, называются **ядрами.**

8.1. Структура процессора

Структуры процессоров отличаются огромным разнообразием от простейшей четырёхбитной до суперскалярной с несколькими конвейерами, одновременно выполняющими десятки команд. Например, на рис. 8.1 изображён процессор, имеющий простую "классическую" структуру.

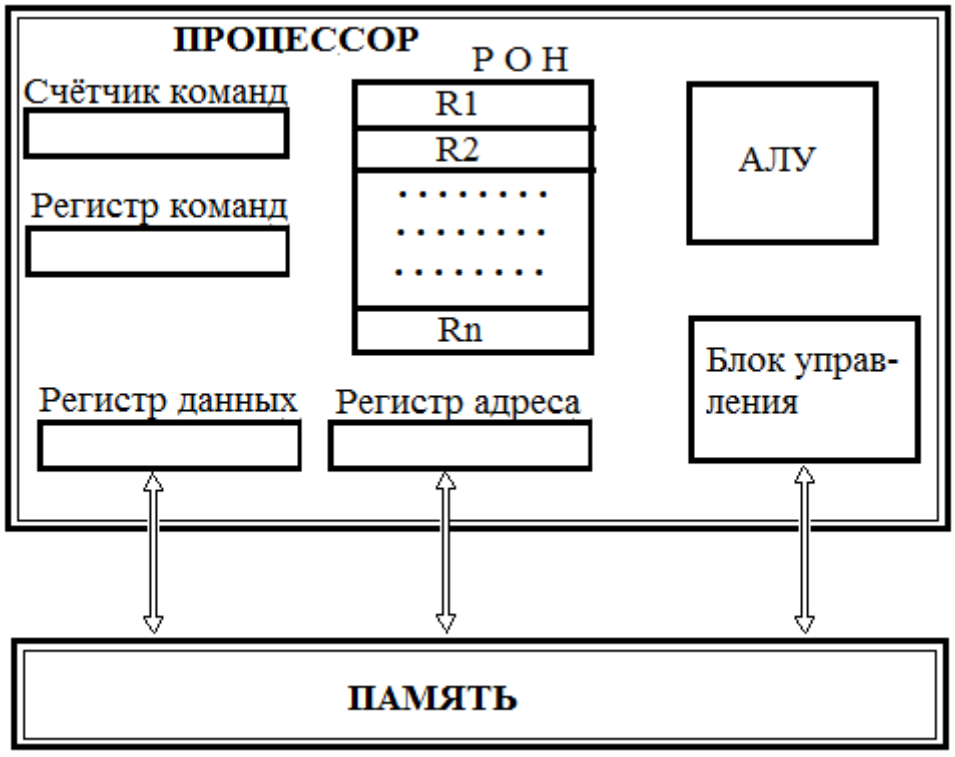


Рис. 8.1 Процессор.

Регистр – функциональное устройство, предназначенное для временного хранения информационного слова (команды, адреса, данных и др.).

РОН - регистры общего назначения - это собственная память процессора, самая быстрая память в ЭВМ. Они доступны программисту. В

РОН хранятся операнды, которые могут потребоваться при выполнении ближайших команд программы. В программе один и тот же операнд часто используется соседними или близко расположенными командами. Необходимость РОН вызвана тем, что процессор работает гораздо быстрее, чем основная память.

АЛУ - арифметико-логическое устройство - служит для выполнения логических и арифметических операций.

Счётчик команд - регистр, играющий центральную роль в организации последовательного выполнения команд.

8.1.2 Последовательность выполнения команды

В разных процессорах за один такт выполняется разное количество действий. Примем для простоты, что за один такт выполняется одна команда. В начале такта на счётчике команд находится адрес команды, которую нужно выполнить. Последовательность действий процессора следующая.

1. Содержимое со счётчика команд передаётся в регистр адреса (**СчКом → РгАдр**).
2. В регистр данных из ячейки ОЗУ с адресом, находящимся в РгАдр, передаются данные в регистр данных (**ОЗУ [РгАдр] → РгДан**).
3. Содержимое счётчика увеличивается на единицу (**СчКом = СчКом +1**).
4. Содержимое регистра данных (команда) передаётся в регистр команд (**РгДан → РгКом**).
5. Выделяется код операции из команды, находящейся в регистре команд.
6. Выполняется выбранная операция.

Действия, перечисленные в пунктах 1,2,4,5, служат для подготовки очередной операции к выполнению. Действие 3 - подготовка к следующему такту. Все первые пять действий для всех команд одинаковые. Шестое действие в зависимости от кода операции выполняется по-разному. Процесс выполнения операции состоит из комбинации следующих действий.

a. Пересылка $\mathbf{Рг_i \rightarrow Рг_j}$ или $\mathbf{Рг_i \leftrightarrow}$ **АЛУ**.
b. Выполнение операции в АЛУ.
c. Передача из ячейки ОЗУ в регистр процессора.
d. Передача из регистра процессора в ячейку ОЗУ.

Особо следует отметить команды условного и безусловного перехода, которые заносят в счётчик команд новый адрес, меняя последовательность выполнения команд.

Для организации управления выполнением команд существуют **аппаратный и микропрограммный** подходы.

При аппаратном подходе для формирования последовательности управляющих сигналов, необходимых для выполнения каждой команды, создаются специальные схемы. Если нужно изменить систему команд, то придётся проектировать новый процессор.

Микропрограммный подход состоит в том, что процесс выполнения команды разбивается на элементарные действия - **микрооперации**. Для каждой команды существует своя последовательность микроопераций, называемая **микропрограммой.**

Микропрограмма хранится в памяти микропрограмм и может быть легко изменена. Этим приёмом пользуются для эмуляции (моделирования) системы команд другой ЭВМ.

Стоимость процессора с микропрограммным управлением ниже, чем с аппаратным.

8.2 Процессорная архитектура

В ходе развития компьютерных технологий были разработаны различные вычислительные системы. Многие из них забыты, а влияние некоторых было весьма значимым. Наметились стратегические тенденции в развитии вычислительной техники и сформировались компьютерные архитектуры. На текущий момент существует несколько основных архитектур и значительное количество процессоров на их основе. **Процессорную архитектуру** можно трактовать как комбинацию вычислительной архитектуры и её реализацию в процессоре, то есть рассматривать в аспекте программирования и аппаратно-технических (и технологических) решений. Нужно отметить, что кардинальное отличие архитектур и их несовместимость обнаруживаются именно на уровне машинного кодирования или низкоуровневого программирования (ассемблирования).

С программной точки зрения, процессорная архитектура определяет набор регистров, команд, их структуру и способ выполнения, в результате чего, с одной стороны, программы, собранные для процессоров одной архитектуры, могут выполняться практически на всех процессорах одинаковой (или подобной) архитектуры, а с другой – не смогут работать

на процессорах иной архитектуры. Для работы на разных платформах производители программного обеспечения вынуждены выпускать специально скомпилированные (или портированные – перенесённые) для них версии.

С аппаратной точки зрения, архитектура процессора – это набор составных частей, компонентов и технологий, присущих линейке процессоров. Аппаратная часть постоянно совершенствуется, как по архитектуре, так и по технологическому процессу. Выпускаются новые поколения процессоров с целью увеличения производительности и функциональности. Так, на рынке существуют процессоры Intel нескольких поколений: Coffee Lake (восьмое поколение), Kaby Lake (седьмое поколение), Skylake (шестое поколение) и другие. Несмотря на смену микроархитектуры (аппаратной части), они остаются программной архитектурой x86, и на них работает всё ранее написанное для этой архитектуры программное обеспечение, за некоторым исключением, если разработчик ПО использовал недокументированные методы, вызовы и процедуры. Поэтому, с точки зрения практического применения процессоров, основной является программная архитектура.

На текущий момент актуальными и распространёнными процессорными архитектурами являются **CISC, RISC, VLIW** архитектуры.

8.2.1 Производительность

Разработчики вычислительных систем постоянно стремятся к повышению их производительности, определяющим показателем которой является количество инструкций (команд), выполненных за единицу времени. Общую формулу производительности можно представить в виде:

$$P = \frac{N}{t},$$

(8.1)

где P – производительность, N – количество инструкций, t – время выполнения. Добавив количество тактов, необходимых для выполнения инструкций - n, формулу (8.1) можно представить следующим образом:

$$\frac{N}{n} \cdot \frac{n}{t}. \qquad (8.2)$$

Первая часть произведения – это количество инструкций, выполняемых за один такт, а вторая – количество тактов процессора за единицу времени, то есть тактовая частота процессора. Таким образом, как следует из выражения (8.2), для увеличения производительности процессора нужно либо поднимать тактовую частоту, либо увеличивать число инструкций, выполняемых за один такт.

Самый простой способ увеличения скорости вычислений – повышение тактовой частоты процессора. Однако на этом пути существуют некоторые технологические ограничения, не позволяющие постоянно наращивать частоту. Поэтому для повышения производительности при данной тактовой частоте процессора применяют параллелизм – исполнение двух и более инструкций одновременно.

Параллелизм может быть на уровне команд и уровне процессоров. В случае параллелизма на уровне команд происходит запуск большего количества команд в секунду. В случае параллелизма на уровне процессоров над одним заданием работают одновременно несколько процессоров. Каждый подход имеет свои преимущества, и в современных системах используются оба подхода.

8.2.2 CISC-архитектура

Первоначально почти все производители первых процессоров использовали архитектуру с расширенным набором команд – **CISC** (**Complex Instruction Set Computer**). Причина этого в том, что разработчики пытались уменьшить так называемый семантический разрыв между тем, что компьютеры способны делать, и тем, что требуют языки программирования высокого уровня, пытаясь заменить одной инструкцией многочисленные машинные коды. Среди микропроцессоров типичными представителями данной архитектуры стали процессоры компании Intel. На начальном этапе развития микропроцессоров (семидесятые годы прошлого века) были и другие процессоры подобной архитектуры производства компаний Motorola, Zilog, MOS Technology и т.д. Но именно благодаря коммерческой привлекательности микропроцессоров Intel эта архитектура стала самой популярной на текущий момент и практически единственной для персональных компьютеров. Даже компания Apple в своих компьютерах Apple Macintosh в итоге перешла от процессоров PowerPC к процессорам Intel.

Первый процессор Intel, обозначивший начало эпохи микропроцессоров, – микросхема Intel 4004 появился в 1971 году [5]. Это был первый коммерческий процессор, реализованный в одной микросхеме. В 1978 году был представлен 16-битный процессор Intel 8086, положивший начало архитектуре x86, или Intel x86. Популярность микроархитектуры x86 была столь велика, что аналогичные процессоры стали выпускать другие производители. В 1985 году компания Intel выпустила первый 32-битный процессор Intel 386. Таким образом сформировалось понятие архитектуры Intel Architecture 32-bit (IA-32), она же Intel x86, или просто

x86 [6]. В дальнейшем она стала 64-битной и получила название x86-64, или AMD64, так как впервые 64-битное расширение архитектуры x86 представила компания AMD. Нужно отметить, что Intel x86 не следует путать с Intel Architecture 64-bit (IA-64), которая является другой архитектуры- **VLIW**. Формально все процессоры x86 являются процессорами CISC-архитектуры.

Итак, x86 – это типичный представитель CISC-архитектуры. Таким образом, в современной интерпретации, говоря CISC, подразумеваем x86, и наоборот.

Для архитектуры CISC характерно:

a. малое количество регистров общего назначения;
b. большое количество различных машинных команд, каждая из которых выполняется за несколько тактов процессора;
c. различные форматы команд с разной длиной;
d. преобладание двухадресной системы команд;
e. развитой механизм адресации операндов.

Основными плюсами данной архитектуры можно считать простоту и эффективность программирования (несколько команд могут быть заменены одной более сложной), а также большое историческое наследие в виде множества написанных для неё программ. Данная архитектура на текущий момент является основной для настольных и серверных систем.

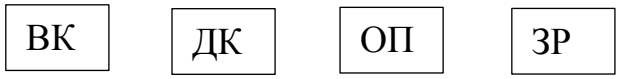
8.2.3. Конвейерная обработка команд

Самый простой способ увеличить быстродействие процессора - увеличить тактовую частоту. Но чем больше тактовая частота, тем больше разогревается интегральная схема процессора. Современные БИС перегреваются при частоте более пяти-шести ГГц. Поэтому рабочую частоту процессора выбирают в полтора-два раза меньше. Больше нельзя. Поэтому основной способ повышения производительности - распараллеливание выполнения программы, т.е. когда процессор при той же тактовой частоте выполняет одновременно несколько последовательных команд. Так как выполнение любой операции состоит из нескольких микроопераций, то можно организовать конвейер, на котором несколько операций находятся на разных стадиях выполнения.

Процедура выполнения команд процессором включает несколько характерных этапов. В простейшем случае можно выделить, как минимум, четыре этапа обработки команд (рис. 8.2, а):

- выборка команды (ВК);
- декодирование команды (ДК);

- выполнение операции (ОП);
- запись результата (ЗР).



а)

Команды n с n+1	ВК	ДК	ОП	ЗР	ВК	ДК	ОП	ЗР
	1	2	3	4	5	6	7	8

номер такта

б)

Команды								
n	ВК	ДК	ОП	ЗР				
n+1		ВК	ДК	ОП	ЗР			
n+2			ВК	ДК	ОП	ЗР		
n+3				ВК	ДК	ОП	ЗР	
	1	2	3	4	5	6	7	8

в) номер такта

Рис. 8.2. Временные диаграммы обработки команд в процессоре:
а) – этапы выполнения команды;
б) – последовательное выполнение команд;
в) – совмещенное выполнение команд (конвейеризация).

Каждый этап в процессоре выполняется за один такт. При последовательной обработке команд (рис. 8.2, б), выполнение следующей (n+1)-й команды начинается только после завершения предыдущей (n)-й команды. Это приводит к низкой производительности и простоям аппаратуры процессора. Для улучшения этих характеристик используется параллельное выполнение нескольких команд путем совмещения в каждом такте различных этапов их обработки (рис. 8.2, в). После выборки n-й команды во 2-м такте идет ее декодирование и выборка (n+1)-й команды. В третьем такте выполняется n-я команда, декодируется (n+2)-я и

осуществляется выборка (n+3)-й команды и т.д. Такая организация работы процессора называется **конвейерной обработкой** (**Pipelining**).

Совмещенные принципы обработки (конвейер команд) существенно увеличивают пропускную способность процессора. Приостановку работы конвейера вызывает любая команда условного перехода в программе или взаимозависимость команд, т.е. использование следующей командой результатов предыдущей команды. Конечно, рассмотренный здесь в качестве примера процессор является гипотетическим. В реальных процессорах конвейер обработки команд сложнее и включает большее количество ступеней. Причина увеличения длины конвейера заключается в том, что многие команды являются довольно сложными и не могут быть выполнены за один такт процессора, особенно при высоких тактовых частотах. Поэтому каждая из четырех (в примере) стадий обработки команд (выборка, декодирование, выполнение и запись) может состоять из нескольких ступеней конвейера. Собственно, длина конвейера – это одна из наиболее значимых характеристик любого процессора. Чем больше длина конвейера, тем большую частоту можно использовать в процессоре.

В реальных процессорах количество ступеней конвейера может быть очень большим. В одной из моделей процессора фирмы Intel конвейер имел 31 ступень, но в следующей модели число ступеней снизилось до 14. Идёт поиск рациональной архитектуры процессора.

Процесс выполнения команды можно разбить на разное количество этапов (микроопераций). Важно, чтобы длительность этапов была приблизительно одинаковой, иначе отдельные блоки конвейера будут простаивать. При параллельно выполнении между командами могут возникнуть конфликты, если:

a. две команды обращаются к одному аппаратному средству;
b. в следующей операции используются результаты предыдущей;
c. выполняется команда условного или безусловного перехода.

Для эффективного использования конвейеров разрабатываются специальные трансляторы, распараллеливающие программу. Для ускорения работы конвейера служит **кеш-память.**

8.2.4 Суперскалярная обработка

Смысл термина «**суперскалярная обработка**» заключается в том, что в аппаратуру процессора закладываются средства, позволяющие одновременно обрабатывать два или более операнда (скаляра), т.е. выполнять две или более команды обработки пары чисел. Суперскалярная

архитектура базируется на многофункциональном параллелизме и позволяет увеличить производительность компьютера пропорционально числу одновременно выполняемых операций. Реализация суперскалярной обработки заключается в чисто аппаратном механизме выборки из буфера инструкций (или кэша инструкций) несвязанных команд и параллельном запуске их на исполнение. На рис. 8.3 приведен конвейер с пятью функциональными блоками.

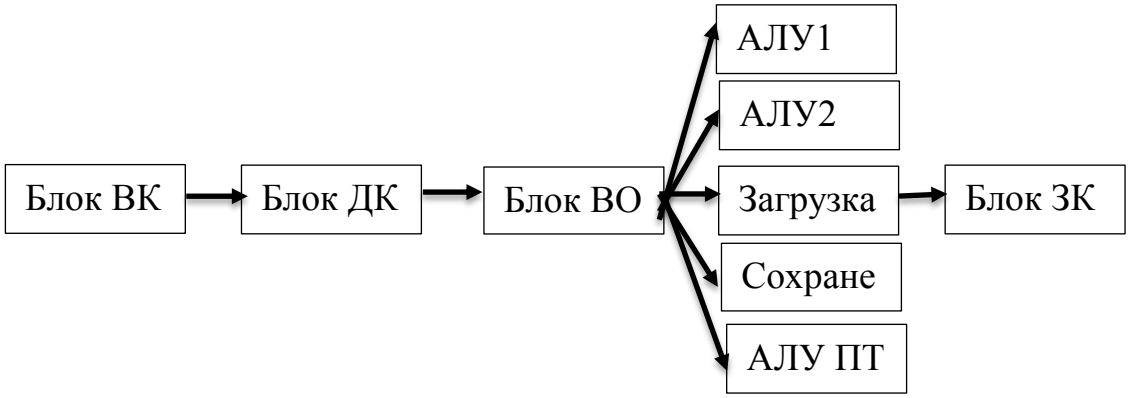


Рис. 8.3

Суперскалярная аппаратура динамически строит план вычислений на основе последовательного кода программы. Хотя такой подход и увеличивает сложность физической реализации, скалярный процессор создает план, используя преимущества тех факторов, которые могут быть определены только во время выполнения.

Этот метод хорош тем, что он «прозрачен» для программиста, составление программ для подобных процессоров не требует никаких специальных усилий, ответственность за параллельное выполнение операций возлагается в основном на аппаратные средства.

Суперскалярная обработка широко используется в современных процессорах корпораций Intel, Advanced Micro Devices (AMD), International Business Machines (IBM), Sun Microsystems/Oracle и др.

8.2.5 RISC-архитектура

Повышение производительности CISC-процессоров из-за особенностей архитектуры приводило к существенному росту количества логических элементов, в результате чего их производство становились всё более сложным и дорогостоящим. Вопросы закрывались конструктивно-технологическими решениями, но в конечном итоге по экономическим соображениям уже не давали адекватного роста производительности. CISC-архитектура в первозданном своём виде достигла потолка производительности. Применение конвейера для повышения

производительности требовало использования простых и быстрых команд. Необходимость дальнейшего роста производительности привела к использованию **RISC архитектуры** (**Reduced Instruction Set Computer**), что означает «компьютер с сокращённым набором команд».

Сравнение CISC и RISC архитектур

	CISC	RISC
Набор команд	Расширенный	Расширенный
Время выполнения команд	Различное	Команда за такт
Программный код	Короткий	Длинный
Использование конвейера	Плохое	Отличное
Реализация процессора	Очень сложная	Простая
Модели процессора	IBM 360/370, DEC PDP/VAX, Intel, AMD, Motorola.	SPARC, PowerPC, ARM, MIPS, МЦСТ R.

Архитектура была разработана в рамках проекта Berkeley RISC. Исследования работы процессоров показали, что реализуемые ими программы попросту не использовали подавляющее большинство инструкций, заложенных в процессоры. Работал принцип 80/20, то есть большее время выполнения типовых программ (примерно 80–90%) приходится на относительную малую часть команд процессора (10–20%). Планировалось создать процессор, который бы содержал лишь самые необходимые инструкции. Естественно, что при этом общее количество процессорных инструкций уменьшилось. Кроме того, принципиальное отличие заключается и в том, что любая инструкция платформы RISC является простой и выполняется за один такт (по крайней мере, должна выполняться), тогда как на выполнение CISC-инструкции могло уходить несколько десятков тактов. При этом длина команды является фиксированной.

Основные особенности RISC-процессоров:

a. Сокращённый набор команд. Первый «настоящий» RISC-процессор имел всего 31 команду. В дальнейшем их количество постепенно росло и достигло 100–200 инструкций в зависимости от реализации процессора. Но это всё равно в несколько раз, а то и на порядок меньше инструкций CISC-процессоров.

b. Большинство команд выполняется за один такт. Все команды выполняются непосредственно аппаратным обеспечением, то есть напрямую без интерпретации микрокомандами. Устранение уровня интерпретации повышает скорость выполнения команд. В компьютерах

типа CISC более сложные команды разбиваются на несколько шагов, которые потом выполняются как последовательность микрокоманд.

c. Большое количество регистров общего назначения. Доступ к памяти происходит относительно медленно. Если слово было загружено из памяти, оно может храниться в регистрах до тех пор, пока не потребуется. Возвращение слова из регистра в память весьма нежелательно, и лучший способ избежать лишних изменений – это наличие достаточного количества регистров.

d. Наличие жёстких многоступенчатых конвейеров. Компьютер должен запускать как можно большее количество команд в секунду. Важным фактором повышения производительности является параллелизм, поскольку запустить на выполнение большое количество команд за короткий промежуток времени можно только в случае, если есть возможность одновременного выполнения нескольких команд. Параллелизм на уровне команд (одновременный запуск) обеспечивается многоступенчатыми конвейерами.

e. Все команды имеют простой формат, и используются немногие способы адресации. Команды легко декодируются, и к памяти обращаются только команды загрузки и сохранения.

f. Наличие вместительной раздельной кэш-памяти. Это необходимо для уменьшения обращений к памяти и тем самым обеспечения необходимого быстродействия для заполнения регистров и конвейеров.

g. Использование оптимизирующих компиляторов, которые анализируют исходный код и частично меняют порядок следования команд.

Упрощение набора команд призвано сократить конвейер, что позволяет избежать задержек на операциях условных и безусловных переходов. Однородный набор регистров упрощает работу компилятора при оптимизации исполняемого программного кода. Кроме того, RISC-процессоры отличаются меньшим энергопотреблением и тепловыделением.

Следует заметить, что количество команд у RISC-процессоров бывает даже больше, чем у CISC. Поэтому слово "сокращённый" вводит в заблуждение и лучше было бы переводить **reduced instruction set computer** как компьютер с упрощённым набором команд, или ещё точнее **компьютер с набором команд упрощённой структуры**.

Уже первые микропроцессоры RISC значительно опережали процессоры CISC по производительности. Учитывая это, можно было предположить, что они должны были занять доминирующее положение на рынке. Но

этого не произошло, по крайней мере, по двум причинам. Во-первых, компьютеры RISC несовместимы с архитектурой Intel x86, а многие компании уже вложили значительные средства в программное обеспечение для продукции Intel. И во-вторых, компания Intel сумела воплотить те же идеи в своей архитектуре. Здесь следует отметить, что упомянутый ранее процессор Intel 386 был последним процессором Intel с так называемой классической CISC-архитектурой.

Столкнувшись с ограничениями по повышению производительности, компания Intel в процессоре следующего поколения Intel 486 применила RISC-ядро и добавила другие элементы RISC-архитектуры, такие как кэш-память и конвейеры. Теперь процессорное ядро стало выполнять самые простые (и обычно самые распространённые) 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 смешанном подходе производительность ниже, чем в архитектуре RISC, новая архитектура CISC имеет ряд преимуществ, поскольку, с одной стороны, появилась возможность повышения производительности, а с другой, можно использовать старое программное обеспечение без изменений. Эта гибридная структура привела современные процессоры x86 к тому, что большую часть элементов процессора занимают элементы, предназначенные для переделки CISC-инструкций в RISC-инструкции, разрешения конфликтов, прогнозирования переходов, исправления последствий неправильных прогнозов и для решения других подобных задач. Для реальной вычислительной работы остаётся только небольшое количество элементов.

Реализацией RISC-архитектуры являются процессоры ARM, MIPS, PowerPC, SPARC и R1000 – российский процессор с 64-битной архитектурой SPARC v.9 производства АО «МЦСТ».

8.2.6 VLIW-Архитектура

Архитектура **VLIW** (**Very Long Instruction Word** - сверхдлинное командное слово) основывается на явно выраженном параллелизме вычислений, заложенном в систему команд процессора - **EPIC** (**Explicitly Parallel Instruction Computing** – вычисления с явным параллелизмом команд). Основной принцип организации этой архитектуры сводится к тому, чтобы перенести нагрузку с периода исполнения в период компиляции. Суперскалярный процессор в ходе исполнения переупорядочивает команды, подменяет регистры, распределяет

функциональные блоки и выполняет множество других функций, что ведёт к максимальной загрузке аппаратных ресурсов. В архитектуре VLIW эти задачи заранее решает компилятор, который располагает полной и достоверной информацией о регистрах процессора и генерирует оптимальный код, в котором нет конфликтов между регистрами. Кроме того, компилятор следит за загрузкой функциональных блоков и не запускает команды, в которых предполагается обращение к занятым функциональным блокам.

Т.е. идея VLIW архитектуры базируется на том, что задача эффективного планирования параллельного выполнения нескольких команд возлагается на «разумный» компилятор. Такой компилятор вначале исследует исходную программу с целью обнаружить все команды, которые могут быть выполнены одновременно, причем так, чтобы это не приводило к возникновению конфликтов. В процессе анализа компилятор может даже частично имитировать выполнение рассматриваемой программы. На следующем этапе компилятор пытается объединить такие команды в пакеты, каждый из которых рассматривается так одна сверхдлинная команда.

Объединение нескольких простых команд в одну сверхдлинную производится по следующим правилам:

- количество простых команд, объединяемых в одну команду сверхбольшой длины, равно числу имеющихся в процессоре функциональных (исполнительных) блоков (ФБ);
- в сверхдлинную команду входят только такие простые команды, которые исполняются разными ФБ, то есть обеспечивается одновременное исполнение всех составляющих сверхдлинной команды.

Длина сверхдлинной команды обычно составляет от 256 до 1024 бит. Такая метакоманда содержит несколько полей (по числу образующих ее простых команд), каждое из которых описывает операцию для конкретного функционального блока.

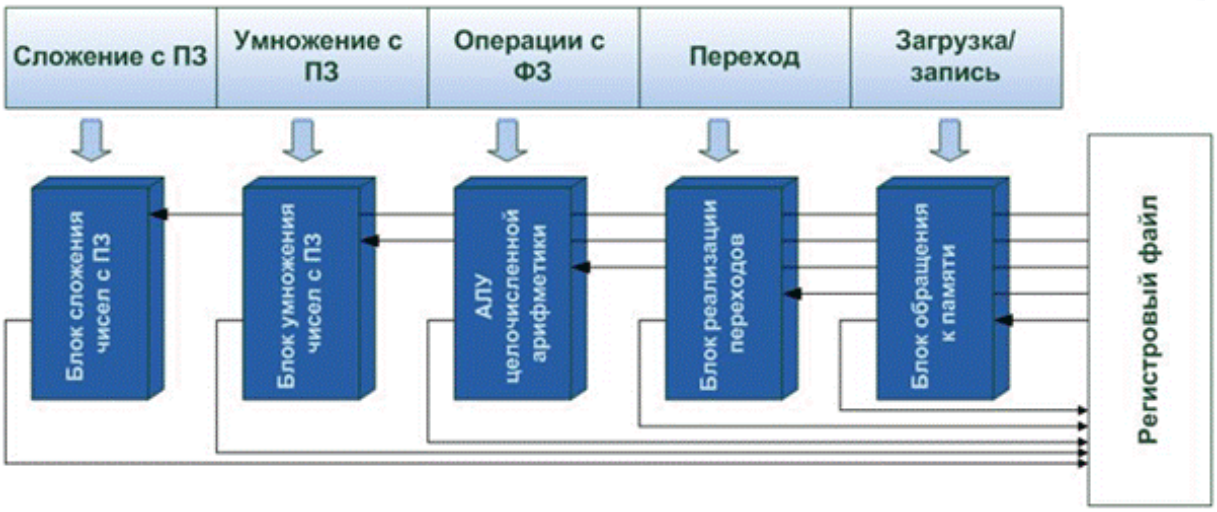


Рис. 8.4

Сказанное иллюстрирует рис 8.4, где показан возможный формат сверхдлинной команды и взаимосвязь между ее полями и ФБ, реализующими отдельные операции.

Как видно из рисунка, каждое поле сверхдлинной команды отображается на свой функциональный блок, что позволяет получить максимальную отдачу от аппаратуры блока исполнения команд.

VLIW-архитектуру можно рассматривать как статическую суперскалярную архитектуру. Имеется в виду, что распараллеливание кода производится на этапе компиляции, а не динамически во время исполнения. То, что в выполняемой сверхдлинной команде исключена возможность конфликтов, позволяет предельно упростить аппаратуру VLIW-процессора и, как следствие, добиться более высокого быстродействия.

В качестве простых команд, образующих сверхдлинную, обычно используются команды RISC-типа, поэтому архитектуру VLIW иногда называют постRISC-архитектурой. Максимальное число полей в сверхдлинной команде равно числу вычислительных устройств и обычно колеблется в диапазоне от 3 до 20. Все вычислительные устройства имеют доступ к данным, хранящимся в едином многопортовом регистровом файле. Отсутствие сложных аппаратных механизмов, характерных для суперскалярных процессоров (предсказание переходов, внеочередное исполнение и т. д.), дает значительный выигрыш в быстродействии и возможность более эффективно использовать площадь кристалла.

Таким образом, в отличие от суперскалярной архитектуры VLIW-процессор имеет простой, а значит, и более быстрый исполнительный конвейер с минимальным количеством ступеней. Итогом такого подхода, с одной стороны, являются следующие преимущества архитектуры:

- более тщательное планирование выполнения программы и оптимизация кода, что даёт лучшее заполнение исполнительных устройств и выполнение большего количества операций за такт;
- в связи с переносом функциональности на компилятор в процессоре меньше места тратится на функциональные блоки, отвечающие за управление, и больше площади остаётся непосредственно на вычислительные ресурсы, такие как регистры, исполнительные устройства и кэш-память;
- это, в свою очередь, приводит к упрощению конструкции процессора и технологического процесса его производства, уменьшению количества транзисторов и понижению тепловыделения.

С другой стороны, необходимо учитывать некоторые особенности:

- сложный компилятор, разработка, поддержка и оптимизация которого довольно ресурсоёмки, требует постоянных доработок и оптимизации и целиком лежит в зоне ответственности разработчика процессора;
- компилятор должен иметь точную информацию об архитектуре процессора – регистрах и исполнительных устройствах. Таким образом, возникает необходимость компиляции кода для каждого типа процессора при изменении в его архитектуре;
- возрастает время компиляции, что, в принципе, не является проблемой при наличии высокопроизводительных рабочих станций у программистов;
- становится сложно учесть динамику исполнения программы при наличии условных ветвлений на основе входящих динамических данных.

Основными сферами применения технологии VLIW-являются цифровые сигнальные процессоры и вычислительные системы, ориентированные на архитектуру IA-64. Наиболее известной была VLIW-система фирмы Multiflow Computer, Inc. (Уже не существующей.) В России VLIW-концепция была реализована в суперкомпьютере Эльбрус 3-1 и получила дальнейшее развитие в его последователе - Эльбрус-2000 (E2к). К VLIW можно причислить семейство сигнальных процессоров TMS320C6x фирмы Texas Instruments. С 1986 года ведутся исследования VLIW-архитектуры в IBM. В начале 2000 года фирма Transmeta заявила процессор Crusoe, представляющий собой программно-аппаратный комплекс. В нем команды микропроцессоров серии x86 транслируются в слова VLIW длиной 64 или 128 бит. Оттранслированные команды хранятся

в кэш-памяти, а трансляция при многократном их использовании производится только один раз. Ядро процессора исполняет элементы кода в строгой последовательности.

Развитие процессорных архитектур характеризуется постоянным стремлением к повышению производительности вычислительных систем. Каждая из указанных архитектур стремится компенсировать недостатки и ограничения других. И в то же время при улучшении одних характеристик могут ухудшиться другие. Поэтому все современные архитектуры находят применение в компьютерных системах, в зависимости от решаемых задач и условий применения.

8.3 Структура команд и адресация

Как уже отмечалось, обработка информации в ЭВМ осуществляется автоматически путем программного управления. Программа представляет собой алгоритм обработки информации, записанный в виде последовательности **команд,** которые должны быть выполнены машиной для получения решения задачи.

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

Процесс выполнения программы состоит из отдельных машинных операций. В данном случае под операцией понимается преобразование информации, выполняемое машиной под воздействием одной команды. Содержанием машинной операции могут быть запись в памяти, передача, арифметическое или логическое преобразование машинных слов и другие процедуры.

По характеру выполняемых операций различают следующие основные группы команд:

1. команды арифметических операций для чисел с фиксированной и плавающей запятой;
2. команды десятичной арифметики;
3. команды логических операций;
4. команды передачи кодов;
5. команды ввода-вывода;
6. команды передачи управления;

7. команды задания режима работы машины.

В команде, как правило, содержатся не сами операнды, а информация об адресах ячеек памяти или регистрах, в которых они находятся.

Код команды состоит из нескольких частей или полей, имеющих определенное функциональное назначение при кодировании командной информации. Команда в общем случае состоит из **операционной** и **адресной** частей. В свою очередь, эти части могут состоять из нескольких полей.

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

Структура команды определяется составом, назначением и расположением полей в команде.

Форматом команды называют ее структуру с разметкой номеров разрядов (битов), определяющих границы отдельных полей команды, или с указанием числа битов в определенных полях.

Важной и сложной проблемой при проектировании ЭВМ является выбор структуры и форматов команды, т.е. ее длины, назначения и размерности отдельных ее полей. Естественно стремление разместить в команде в возможно более полной форме информацию о предписываемой командой операции. Однако в условиях, когда в современных ЭВМ значительно возросло число выполняемых различных операций и соответственно команд (например, в системе команд x86 более 500 команд), а также значительно увеличилась емкость адресуемой основной памяти (сотни Гбайт), это приводит к недопустимо большой длине формата команды.

Число двоичных разрядов, отводимых под код операции, должно быть таким, чтобы можно было представить все выполняемые машинные операции. Если ЭВМ выполняет N различных операций, то число разрядов в коде операции $n_{\text{коп}} \geq \log_2 N$.

Например, при N= 600 $n_{\text{коп}}$=10.

Если основная память содержит S адресуемых ячеек (байтов), то для явного представления только одного адреса необходимо в команде иметь адресное поле одного операнда с числом разрядов $n_{\text{А}} \geq \log_2 S$.

Например, при S=64 Гбайт $n_{\text{А}}$ = 36.

Отмечавшиеся ранее характерные для процесса развития ЭВМ расширение системы (наборы) команд и увеличение емкости основной памяти, а особенно создание микроЭВМ с коротким словом, потребовали

разработки методов сокращения длины команды. При решении этой проблемы существенно видоизменилась структура команды, получили развитие различные способы адресации информации.

Чтобы команда содержала в явном виде всю необходимую информацию о задаваемой операции, она должна, как это показано на рис. 8.5, б, содержать следующую информацию:

• A_1, A_2 — адреса операндов;

• A_3 — адрес результата;

• A_4 —адрес следующей команды (принудительная адресация команд). Такая структура приводит к большой длине команды.

Например, при N= 600, S = 4 Гб длина команды составит 154 бит и неприемлема для прямой адресации операндов основной памяти. (Отметим, что в компьютерах с RISC-архитектурой четырехадресные команды используются для адресации операндов, хранящихся в регистровой памяти процессора).

Можно установить, что после выполнения данной команды, расположенной по адресу K (и занимающей L ячеек), выполняется команда из (K +L)-й ячейки. Такой порядок выборки команды называется естественным. Он нарушается только специальными командами передачи управления. В таком случае отпадает необходимость указывать в команде в явном виде адрес следующей команды.



Рис. 8.5 Структуры команд: *а*- обобщенная; *б* - четырехадресная;

в -трехадресная; *г* - двухадресная; *д* - одноадресная; *е* – безадресная.

В одноадресной команде (рис. 8.5, *д*) подразумеваемые адреса имеют уже и результат операции, и один из операндов. Один из операндов указывается адресом в команде, в качестве второго используется содержимое регистра процессора, называемого в этом случае регистром результата или аккумулятором. Результат операции записывается в тот же регистр. Наконец, в некоторых случаях возможно использование безадресных команд (рис. 4.12, *е*), когда подразумеваются адреса обоих операндов и результата операции, например при работе со стековой памятью.

В трехадресной команде (рис. 4.12, *в*) первый и второй адреса указывают ячейки памяти, в которых расположены операнды, а третий определяет ячейку, в которую помещается результат операции. Можно условиться, что результат операции всегда помещается на место одного из операндов, например, первого. Тогда получим двухадресную команду (рис. 8.5, *г*), т.е. для результата используется подразумеваемый адрес.

Следует различать понятия «**адресный код** в команде $А_к$» и «**исполнительный** (физический адрес) адрес $А_и$».

Адресный код - это информация об адресе операнда, содержащаяся в команде.

Исполнительный адрес -это номер ячейки памяти, к которой производится фактическое обращение.

Таким образом, способ адресации можно определить как способ формирования исполнительного адреса операнда $А_и$ по адресному коду команды $А_к$.

В системах команд современных ЭВМ часто предусматривается возможность использования нескольких способов адресации операндов для одной и той же операции. Для указания способа адресации в некоторых системах команд выделяется специальное поле в команде — поле «метод» (указатель адресации, УА), рис. 8.6, *а*.

В этом случае любая операция может выполняться с любым способом адресации, что значительно упрощает программирование.

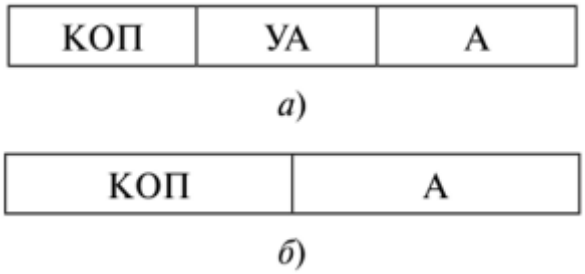


Рис. 8.6. Общая структура команды: *а* - с указателем метода адресации; *б* -без указателя метода адресации

Классификация способов адресации по наличию адресной информации в команде:

- явная адресация;
- неявная адресация;
- относительная.

При явной адресации операнда в команде есть поле адреса этого операнда, в котором задается адресный код $А_к$. Большинство методов адресации являются явными.

При неявной адресации адресное поле в команде отсутствует, адрес операнда подразумевается кодом операции.

Широко используются следующие методы адресации операнда с различной кратностью обращения (r) в память:

1. непосредственная (r=0);
2. прямая (r= 1);
3. косвенная (r> 2).

Непосредственная адресация операнда. Обращение к регистровой памяти (РП) или оперативной памяти (ОП) не производится (r=0). Таким образом уменьшается время выполнения операции, сокращается объем памяти.

Прямая адресация операндов. При этом способе (рис. 8.7.) адресации обращение за операндом в РП или ОП производится по адресному коду в поле команды, т.е. исполнительный адрес операнда совпадает с адресным кодом команды ($А_{исп} = А_к$). Обеспечивая простоту программирования, этот метод имеет существенные недостатки, так как для адресации к ячейкам памяти большой емкости требуется «длинное» адресное поле в команде.

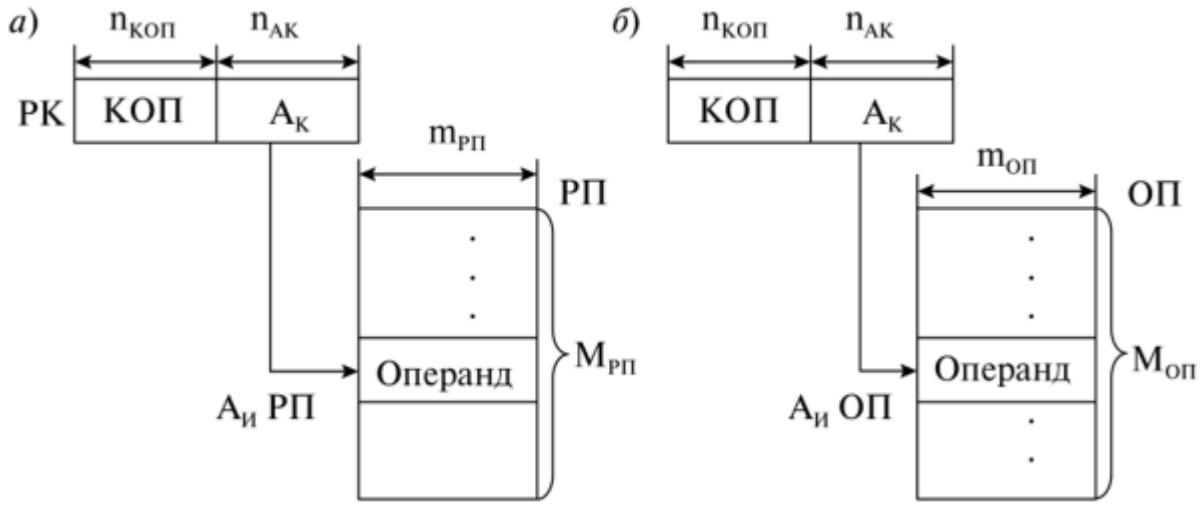


Рис. 8.7. Схема прямой адресации: *а* - к регистровой памяти; *б* - к основной памяти

Косвенная адресация операндов. Адресный код команды указывает адрес ячейки памяти, в которой находится не сам операнд, а лишь адрес операнда, называемый указателем операнда. Адресация к операнду через цепочку указателей (косвенных адресов) называется косвенной.

Адрес указателя остается неизменным, а косвенный адрес может изменяться в процессе выполнения программы. Это обеспечивает переадресацию данных, т.е. упрощает обработку массивов и списковых структур данных, упрощает передачу параметров подпрограммам, но не обеспечивает перемещаемость программ в памяти (рис. 8.8, *а*).

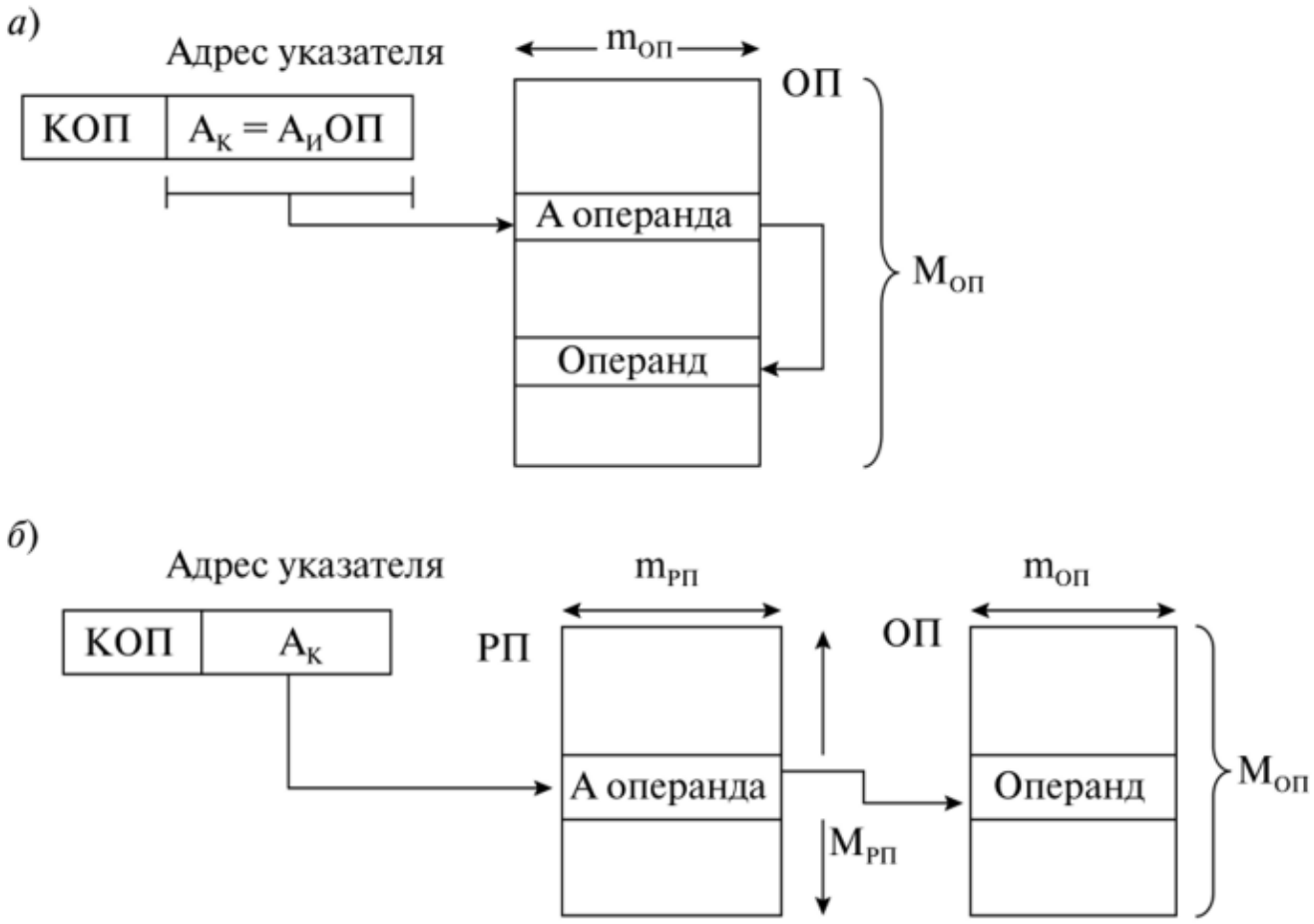


Рис. 8.8. Схема косвенной адресации: *а* -указатель операнда и операнд расположены в одном адресном пространстве оперативной памяти; *б* - указатель операнда расположен в регистровой памяти, а операнд - в адресном пространстве оперативной памяти.

Относительная адресация

Базирование способом суммирования. В команде адресный код $A_к$ разделяется на две составляющие: $A_Б$ -адрес регистра в регистровой памяти, в котором хранится база Б (базовый адрес); С - код смещения относительно базового адреса (рис. 8.9.).

С помощью метода относительной адресации удается получить так называемый перемещаемый программный модуль, который одинаково выполняется процессором независимо от адресов, в которых он расположен. Начальный адрес программного модуля (база) загружается, при входе в модуль, в базовый регистр. Все остальные адреса программного модуля формируются через смещение относительно начального адреса (базы) модуля. Таким образом, одна и та же программа может работать с данными, расположенными в любой области памяти, без

перемещения данных и без изменения текста программы только за счет изменения содержания всего одного базового регистра. Однако время выполнения каждой операции при этом возрастает.

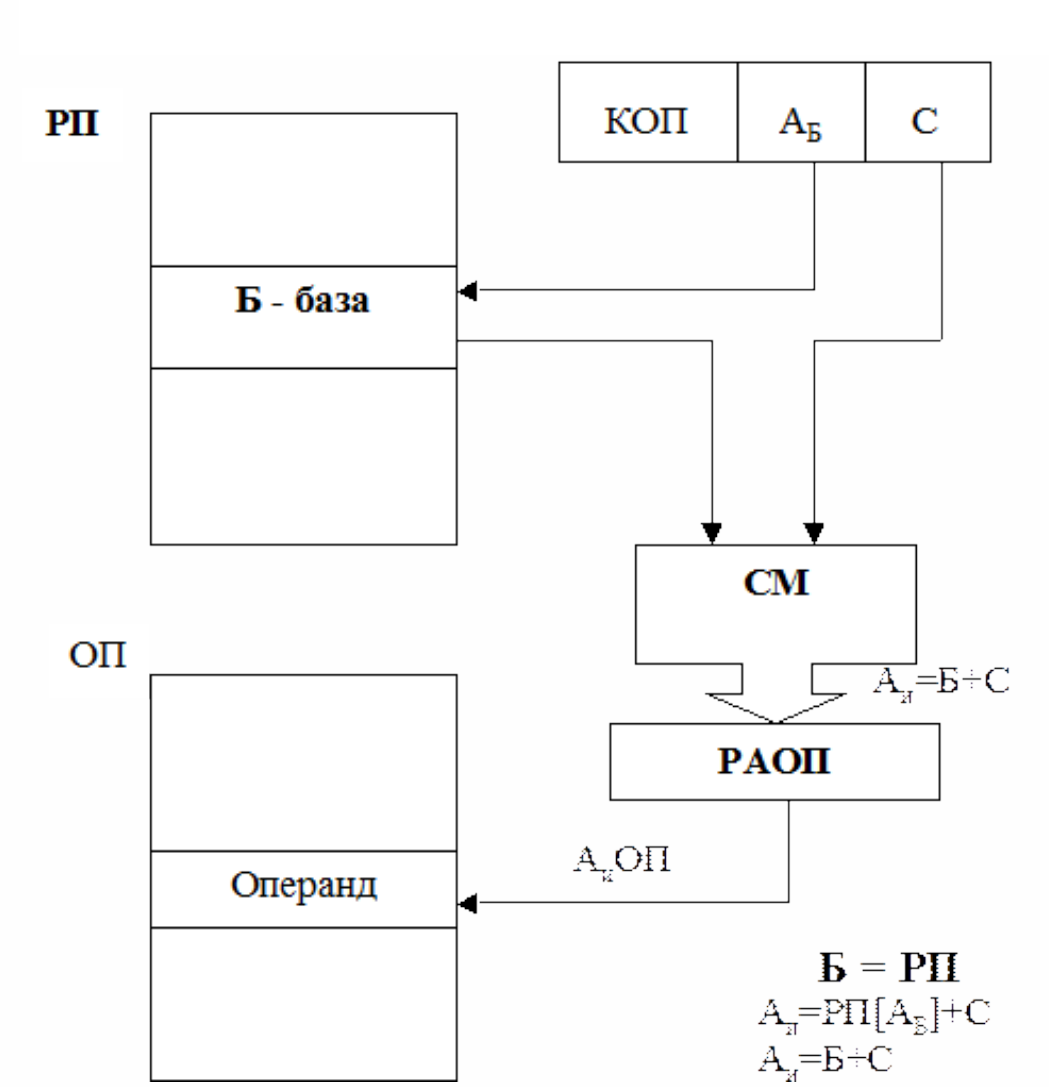


Рис. 8.9. Схема формирования относительного адреса способом суммирования кодов базы и смещения.

СМ - сумматор,

РАОП - регистр адреса ОП,

Б – база (базовый адрес),

С - смещение,

$A_Б$ – адрес регистра базы

Базирование способом совмещения составляющих. Для увеличения емкости адресной ОП без увеличения длины адресного поля команды можно использовать для формирования исполнительного адреса совмещение (конкатенацию) кодов базы и смещения (рис. 8.10.). Однако в данном случае начальные адреса массивов не могут быть реализованы произвольно, а должны иметь в младших разрядах n нулей, где n – длина поля смещения.

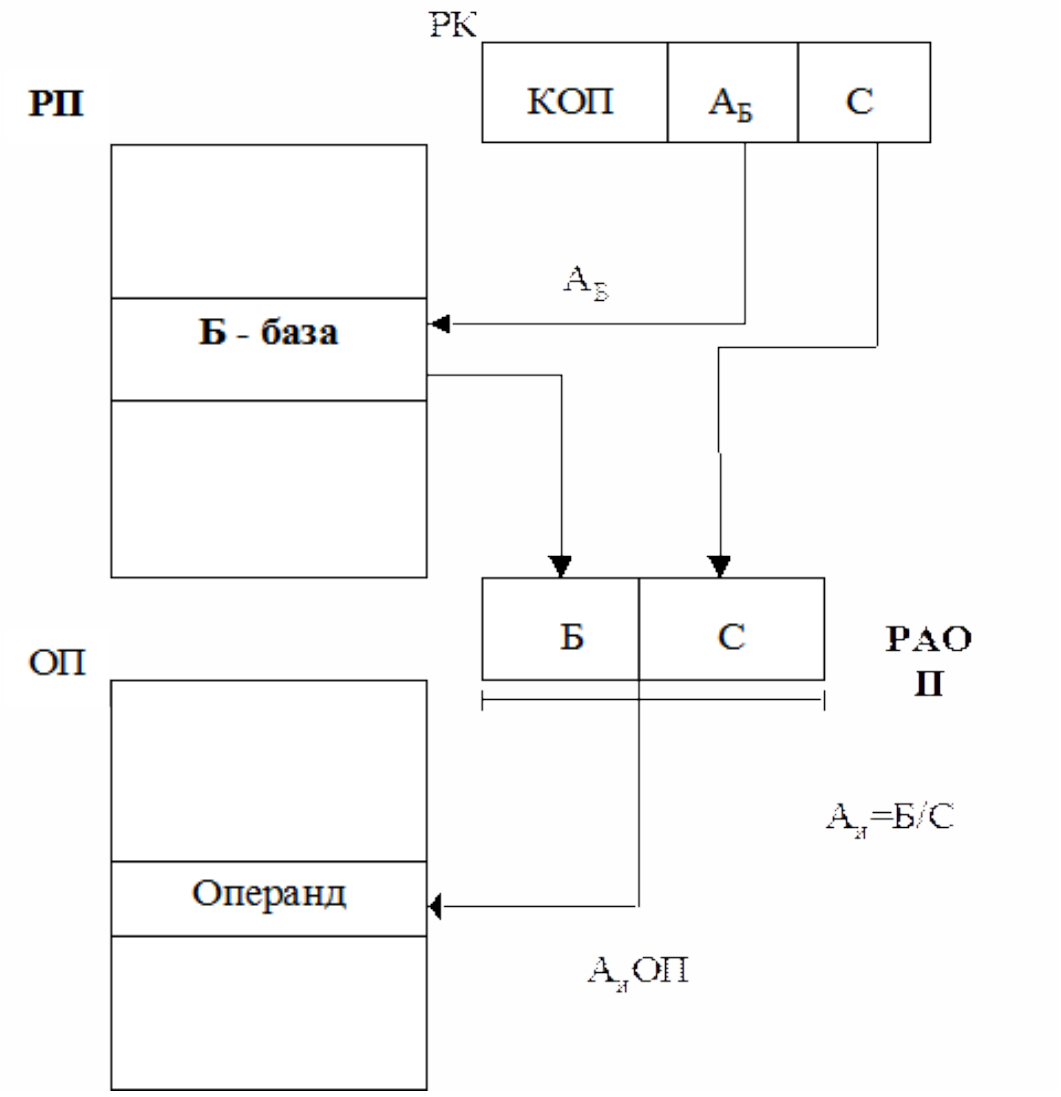


Рис. 8.10. Схема формирования относительного адреса способом совмещения кодов базы и смещения.

Индексная адресация. Для работы программ с массивами, требующими однотипных операций над элементами массива, удобно использовать индексную адресацию. Схема индексной адресации аналогична базированию путем суммирования. В этом случае адрес *i*-го операнда в массиве определяется как сумма начального адреса массива (задаваемого полем смещения С) и индекса И, записанного в одном и регистров РП, называемом теперь **индексным регистром**. Адрес индексного регистра задается в команде полем адреса индекса - $A_{ин}$ (аналогично $A_Б$).

В каждом *i*-м цикле содержимое индексного регистра изменяется на постоянную величину (часто равную 1). Использование индексной адресации значительно упрощает программирование циклических алгоритмов.

Для эффективной работы при относительной адресации применяется комбинированная индексация с базированием, при которой адрес операнда вычисляется как сумма трех величин (рис. 8.11.) $А_и$ = Б + И + С.

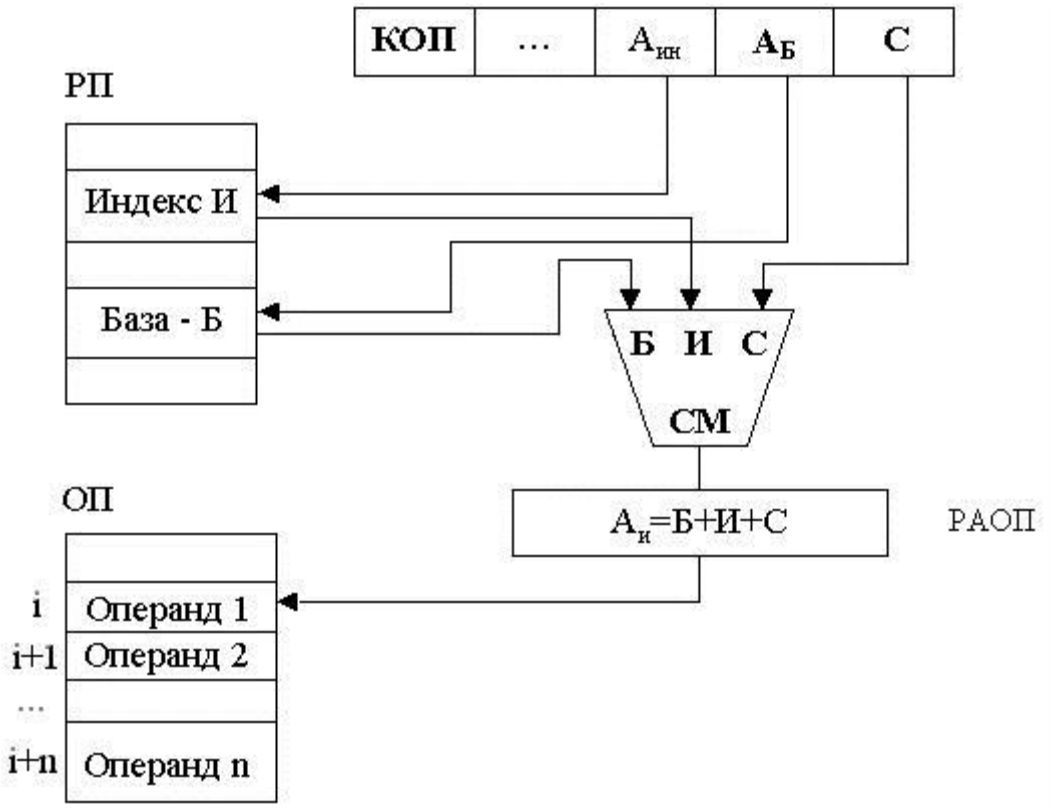


Рис. 8.11. Схема формирования дополнительного адреса при индексной адресации и базировании: $А_{ИН}$ – адрес индексного регистра

8.4 Система прерываний

Как сообщить пользователю, что на принтере кончилась бумага, или запустить по сигналу таймера программу резервного копирования? Для этих целей служит **регистр прерываний**, в котором каждый разряд соответствует своей причине **прерывания.**

Прерывание - это прекращение выполнения текущей команды или текущей последовательности команд для обработки некоторого события специальной программой - обработчиком прерывания, с последующим возвратом к выполнению прерванной программы. Событие может быть вызвано особой ситуацией, сложившейся при выполнении программы, или сигналом от внешнего устройства. Прерывание используется для быстрой реакции процессора на особые ситуации, возникающие при выполнении программы и взаимодействии с внешними устройствами. Механизм

прерывания обеспечивается соответствующими аппаратно-программными средствами компьютера.

Т.е., когда на определённый разряд РП поступает сигнал прерывания, то запускается соответствующая программа обработки. Выполнявшаяся в этот момент программа прерывается. Её состояние (команда, на выполнении которой произошло прерывания, РОНы и т. д.) запоминается и после реакции на прерывание выполнение прерванной программы возобновляется

Назначение: Любая особая ситуация, вызывающая прерывание, сопровождается сигналом, называемым запросом прерывания (ЗП). Запросы прерываний от внешних устройств поступают в процессор по специальным линиям, а запросы, возникающие в процессе выполнения программы, поступают непосредственно изнутри микропроцессора.

Аппаратные прерывания используются для организации взаимодействия с внешними устройствами:

- маскируемые, которые могут быть замаскированы программными средствами компьютера;
- немаскируемые, запрос от которых таким образом замаскирован быть не может.

Программные прерывания вызываются следующими ситуациями:

- (переполнение, нарушение защиты памяти, отсутствие нужной страницы в оперативной памяти и т.п.);
- специальной команды прерывания INT *n*, используемой обычно программистом при обращениях к специальным функциям операционной системы для ввода-вывода информации.

Функции прерываний:

1. определение наиболее приоритетного незамаскированного запроса на прерывание (если одновременно поступило несколько запросов);
2. определение типа выбранного запроса;
3. сохранение текущего состояния счетчика команд и регистра флагов;
4. определение адреса обработчика прерывания по типу прерывания и передача управления первой команде этого обработчика;
5. выполнение программы - обработчика прерывания;
6. восстановление сохраненных значений счетчика команд и регистра флагов прерванной программы;
7. продолжение выполнения прерванной программы.

На рис. 8.12. представлена временная диаграмма выполнения прерывания.

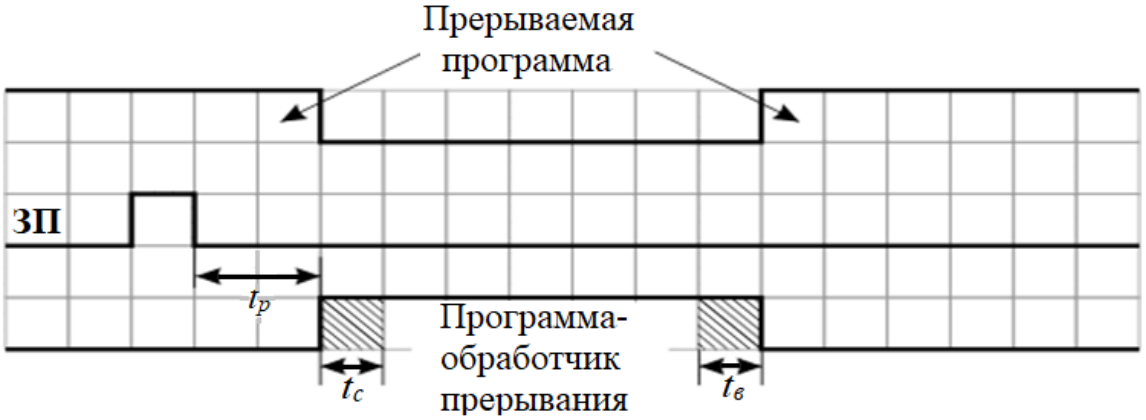


Рис. 8.12. Выполнение прерывания в компьютере: t_p - время реакции процессора на запрос прерывания; t_c - время сохранения состояния прерываемой программы и вызова обработчика прерывания; $t_в$ - время восстановления прерванной программы.

На некоторые сигналы прерывания нужно отреагировать немедленно, например, на сбой оборудования, обслуживание других можно отложить на допустимый промежуток времени. Прерывания могут образовать очередь. Для выбора из очереди кандидата на обслуживание каждому прерыванию присваивается приоритет. Во многих управляющих комплексах существует очень сложная система приоритетов. В простейшем случае используются три вида приоритетов:

1. обслуживание без приоритетов, в свою очередь имеющее три разновидности:
    a. первым пришёл, первым обслужился (**FIFO**);
    b. последним пришёл, первым обслужился (**LIFO**);
    c. в случайной последовательности.
2. Обслуживание с относительными приоритетами - обслуживаемая заявка (выполняющаяся программа) не прерывается, выполняется до конца, после чего начинается выполнение заявки с самым высоким приоритетом;
3. Обслуживание с абсолютными приоритетами - обслуживаемая заявка прерывается заявкой с самым высоким приоритетом.

Лекция 9.

Шины

Шина в ЭВМ – это обычный токопроводящий кабель или система проводников на печатной плате.

Назначение шины - передача информации между устройствами ЭВМ. Шина состоит из линий трёх типов:

- шина данных;
- адресная шина;
- шина управления.

Сложная шина может состоять из более простых шин. На рис. 9.1 шина, связывающая процессор и память разделена на шину данных, шину адреса и шину управления. Данные по шине передаются в определённой последовательности, по определённым правилам.

Набор правил, полностью регламентирующий передачу данных по шине, называется протоколом.

Шина входит в состав ***интерфейса*** - общего понятия, служащего для описания взаимодействия устройств вычислительной системы.

Интерфейс - совокупность аппаратных и программных средств и правил обеспечивающих взаимодействие устройств вычислительной системы.

В современных ЭВМ часто применяется общая шина, принцип работы которой заключается в следующем:

- все подключённые к шине устройства равноправны;
- в каждой передаче данных одно из устройств инициирует передачу и называется хозяином (Master), другое устройство называется подчинённым или ведомым (Slave);
- инициация - это передача по шине хозяином адреса ведомого;
- все устройства дешифрируют выставленный адрес, одно (Slave) опознаёт себя и между Master и Slave происходит передача данных.

Шины могут быть синхронными, передающими данных только по тактовым импульсам, и асинхронными, осуществляющими передачу данных в произвольные моменты времени. Синхронные шины имеют большую скорость передачи, чем сравнимые с ними асинхронные, но требуют тщательной разработки.

На рис. 9.1 изображена архитектура типичной ПЭВМ. К ***шине процессора*** подключаются устройства, которым требуется очень высокая скорость взаимодействия с процессором.

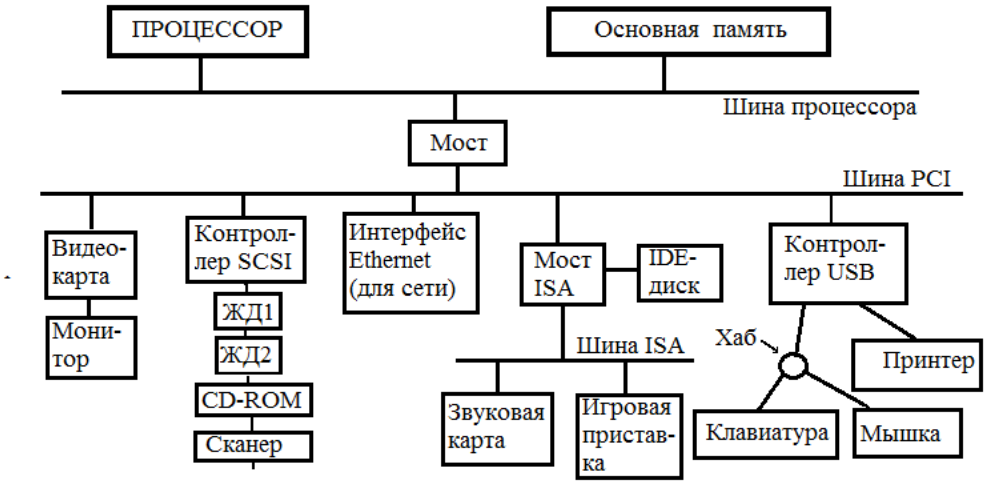


Рис. 9.1. Архитектура типичной ПЭВМ

Через неё с процессором связана основная память (ОЗУ). Так как к шине процессора непосредственно не может быть подсоединено большое количество устройств, то с ней через мост связана ***шина расширения***. Мост вызывает небольшую задержку во взаимодействии процессора с подключёнными к шине расширения устройствами.

В примере на рис. 9.1 в качестве шины расширения использована шина ***PCI*** (***Peripheral component interconnect***, дословно - взаимосвязь периферийных компонентов). Она поддерживает типичные для шины процессора функции в стандартизированном формате независимо от типа процессора. В ней используется технология ***plug-and-play*** (включил и работай, дословно, играй). Устройство, подключаемое к компьютеру через шину PCI, должно иметь постоянную память, в которой содержится вся необходимая информация для автоматического подключения этого устройства.

SCSI-шина, в отличие от жестких шин расширения, реализуется в виде отдельного кабельного шлейфа, который допускает соединение до 8 устройств (спецификация SCSI-1) внутреннего и внешнего исполнения. Применяется SCSI-шина преимущественно в серверах.

ISA (***Industry Standard Architecture***) — 8- или 16-разрядная шина ввода-вывода IBM PC-совместимых компьютеров. Служит для подключения плат расширения стандарта ISA.

USB (***Universal Serial Bus*** — «универсальная последовательная шина») - последовательный интерфейс для подключения периферийных (находящихся вне материнской платы) устройств. USB-шина имеет древовидную структуру (рис.9.2)

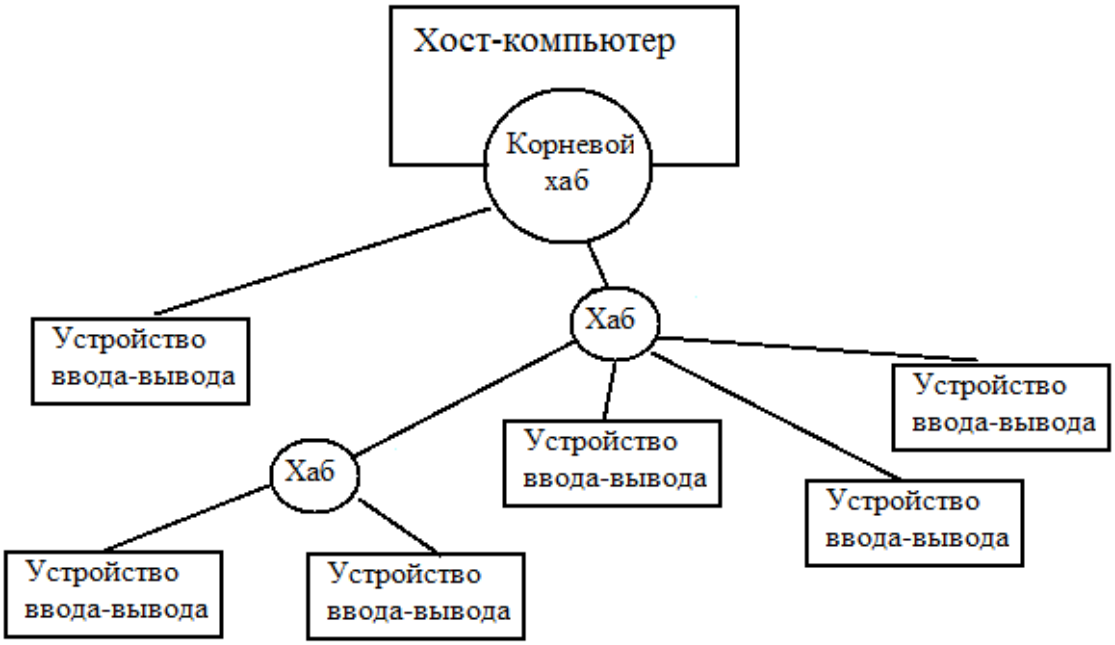


Рис. 9.2. Структура шины USB

В каждом узле дерева находится специальное устройство — ***концентратор*** (***хаб***). Хаб занимается электрическим согласованием кабелей, маршрутизацией пакетов, обнаружением подключения/отключения устройств и другими функциями. Хаб имеет несколько входов (портов) для подключения устройств и других хабов.

Портом называется вход-выход устройства. Порты стандартизированы и согласуются со стандартами шин. Порт состоит из разъёма, к которому подсоединяется устройство, и схем согласования шины, ведущей от устройства, с шиной компьютера.

Кабель USB 2.0 содержит в себе четыре линии — пару для приёма/передачи данных, плюс и ноль питания, разъём имеет 4 контакта. Для передачи высокоскоростных ***SuperSpeed*** сигналов в USB 3.0 добавлено ещё четыре линии связи (две витые пары) и один контакт сигнальной земли (***GND_DRAIN***). Новые контакты в разъёмах USB 3.0 расположены отдельно от старых в другом контактном ряду.

USB-2 имеет скорость передачи 480 Мбит/сек., а USB-3 - 5 Гбит/сек.

Разъёмы USB устроены так, что сначала соединяются линии заземления, а затем - остальные линии шины. Это позволяет подключать к компьютеру устройства, не выключая питания. Удобные, компактные разъёмы, высокая скорость передачи данных, возможность быстро подключать и отключать большое количество устройства сделали USB самой популярной внешней шиной.

9.1 Шина PCI Express

PCI Express - компьютерная шина, использующая программную модель шины PCI и высокопроизводительный физический протокол, основанный на последовательной передаче данных. Коммутатор и устройства имеют соединение (рис. 9.3) "точка-точка", т. е. соединяются непосредственно друг с другом в отличие от соединения через общую шину, когда множество устройств подсоединяются к общему кабелю. Поэтому физически PCI Express шиной не является.

Строго говоря, соединение с коммутатором - двухточечное. Оно состоит из двух однонаправленных каналов (линий), по одному в каждом из направлений. Канал состоит из двух проводов (сигнального и заземляющего), что обеспечивает высокую помехозащищённость. При таком соединении быстрое устройство не зависит от медленного, что было при соединении с общей шиной.

Соединение этого типа произошло от соединения "звезда", применяемого в компьютерных сетях. Оно предполагает, в отличие от общей шины, использование дополнительного оборудования (коммутатор), но значительно увеличивает надёжность и упрощает взаимодействие устройств.

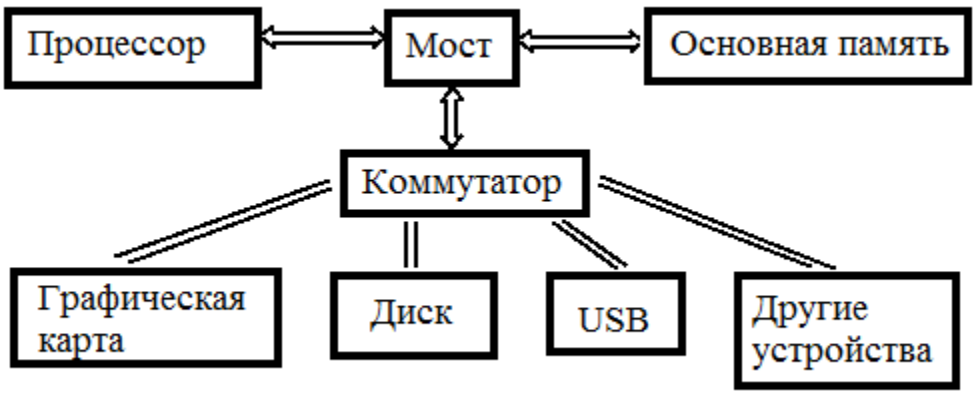


Рис. 9.3. Шина PCI Express

Данные через PCI Express передаются не по одному слову, а пакетами. Пакет содержит заголовок, в котором передаётся управляющая информация. Отпадает необходимость в управляющих сигналах и управляющей линии, но усложняется протокол передачи данных.

9.2 Прямой доступ к памяти

Как отмечалось в лекции 6, обмен данными между оперативной памятью и внешними устройствами происходит не по одному слову, а сразу целыми блоками последовательно расположенных в памяти слов. В

первых ЭВМ процессор после окончания передачи очередного слова сообщал контроллеру ОЗУ адрес следующего слова. В современных ЭВМ процессор перед началом передачи блока слов сразу сообщает контроллеру внешнего устройства размер передаваемого блока слов и адрес первого слова в пакете. После этого процессор в передаче блока участия не принимает, а управляет передачей блока контроллер. Таким образом, внешнее устройство взаимодействует с памятью напрямую, минуя процессор. После окончания передачи блока процессор получает сигнал через прерывание. Такой способ передачи получил название ***прямой доступ к памяти (ПДП или DMA - Direct Memory Access)***, а соответствующий контроллер - ***контроллер ПДП.***

Конструкторы стремятся повысить производительность ЭВМ за счёт одновременной работы как можно большего количества устройств. В одном процессоре одновременно обрабатывают информацию несколько конвейеров. В многопроцессорной ЭВМ одновременно работают несколько процессоров. ПДП - это способ одновременной работы нескольких внешних устройств с памятью. Чтобы одновременно загрузить множество устройств ЭВМ, понадобилось создать новый вид программ - операционную систему, с помощью которой организуется ***мультипрограммный режим работы ЭВМ.***

Лекция 10.

Память ЭВМ

Необходимый и доступный объём памяти

Современное программное обеспечение требует огромных объёмов памяти. Например, ПЭВМ с операционной системой Windows 7 (домашняя 32-разрядная версия) занимает 29 Гбайт, антивирус - 1 Гбайт. Очень много места требуют для хранения фотографии. Еще более значительных объемов памяти требуют фильмы. Один средний фильм занимает несколько Гбайт памяти. Тем не менее, современные магнитные диски ёмкостью до восьми Тбайт вполне удовлетворяют большинство пользователей ПЭВМ. Правительственные организации, корпорации и научно-исследовательские центры могут позволить себе иметь пакеты магнитных дисков с практически неограниченной суммарной ёмкостью. Для этих организаций актуальна другая проблема; как обработать за приемлемое время накопленную информацию.

Виды памяти

В предыдущих лекциях были рассмотрели регистры как элементы памяти, в которых можно не только хранить, но и обрабатывать информацию. Именно эта особенность выделяет регистры в особый вид памяти. Регистры - самый быстрый, но и самый дорогой вид памяти. Из-за дороговизны их объём в ЭВМ ничтожен по сравнению с общим объёмом памяти. В некоторых процессорах имеется по 128 регистров для хранения чисел с фиксированной и чисел с плавающей запятой. Обычно объём регистровой памяти ещё меньше.

По способу поиска данных память делят на две группы: **память с пронумерованными ячейками**, т.е. с адресами, и **ассоциативную память**. В ассоциативной памяти информацию ищут не по адресу, а по признаку. Самый простой пример ассоциативной памяти - словарь. Аппаратным путём из ассоциативной памяти выбирается перевод заданного слова. Другой пример, в связной ЭВМ сообщение ищется по его заголовку. Ассоциативной является к**еш-память** первого уровня. Ассоциативная память также дорогая, но её объём может доходить до нескольких Мбайт.

Наиболее просто устроена память, состоящая из ячеек, каждая из которых имеет свой адрес. В современных ЭВМ длина адресуемой ячейки (разрядность), как правило, составляет 1 байт. Такое устройство имеет оперативная память (**ОЗУ** - оперативное запоминающее устройство).

Проблема адресации

В реальной ЭВМ объём памяти - несколько Гбайт и при длине машинной команды в 32 бита даже один адрес в неё не поместится. Зависимость возможного объёма адресуемой памяти от длины адреса представлена в табл. 10.1.

Таблица 10.1. Зависимость объёма адресуемой памяти от длины адреса

Длина адреса (байт)	16	32	40
Объём адресуемой памяти	2^{16} байт 64 Кбайт	4 Гбайт $4*10^9$ байт	1 Тбайт $4*10^{12}$ байт

Проблема длинного адреса решается путём использования косвенной адресации.

Организация связи процессора с памятью

Пересылка данных между процессором и памятью осуществляется с помощью двух регистров процессора (рис. 10.1) - регистра адреса и регистра данных. Операция пересылки начинается с загрузки адреса ячейки памяти на регистр адреса.

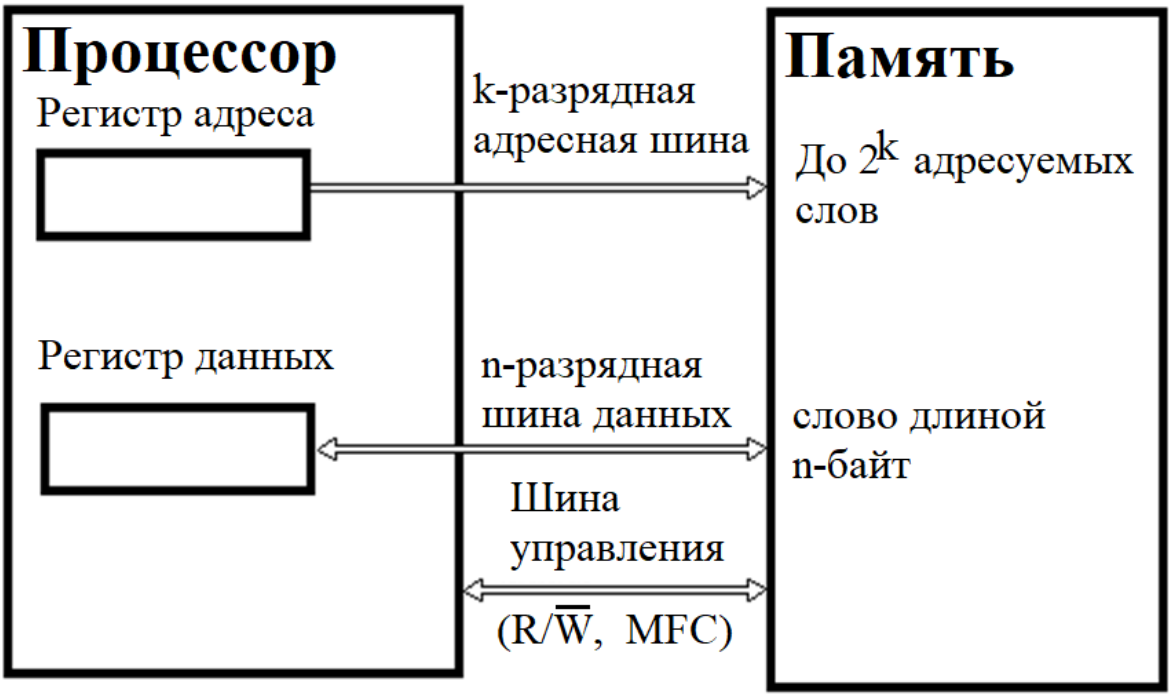


Рис. 10.1. Связь процессора с памятью
R/W - сигнал чтения/записи
MFC - сигнал завершения операции

Если нужно выполнить операцию чтения, процессор устанавливает управляющую линию $\boldsymbol{R/\overline{W}}$ в состояние 1. В ответ память помещает данные на шину данных и выдаёт сигнал **MFC** (**Memory Function Completed** - операция памяти завершена). Приняв этот сигнал, процессор загружает данные с шины данных в регистр данных.

Чтобы выполнить операцию записи в память, процессор загружает передаваемые данные в регистр данных, а адрес - в регистр адреса и устанавливает управляющую линию $\boldsymbol{R/\overline{W}}$ в состояние 0.

Разрядность шины адреса (т.е. количество линий в шине адреса) определяют объём адресуемой памяти, которая составляет $V = 2^k$ ячеек. Общепринятая длина ячейки памяти - 1 байт.

Количество разрядов *n* регистра данных и в шине адреса равны длине ячеек памяти.

В реальных ЭВМ длина машинного слова - величина переменная, но обмен данными с памятью производится побайтно.

Быстродействие памяти

Для измерения быстродействия памяти используются две характеристики:

- **время доступа к памяти**, определяемое как время, прошедшее между моментом инициирования операции и моментом её завершения, например, между сигналом чтения и MFC;
- **время цикла** - время между моментами начала двух последовательных операций.

Время цикла несколько больше времени доступа к памяти.

Классификация памяти

В начале лекции были выделены три вида памяти: регистровая, ассоциативная и память, состоящая из нумерованных ячеек фиксированной длины. Физически все эти виды памяти - полупроводниковые. По времени доступа память делится на два вида.

- **Память с произвольным доступом** - память, в которой на обращение по любому адресу всегда уходит одинаковое время. К ней относится вся память на полупроводниковых интегральных схемах.
- **Память с последовательным или полупоследовательным доступом** - память, в которой время обращения зависит от состояния устройства. К ней относится электромеханическая память, т. е. память с механическими частями (магнитные и оптические диски, магнитные ленты).

Физические типы памяти

По физическим принципам работы запоминающие устройства делятся на полупроводниковые и электромеханические (рис.10.2). До недавнего времени вся внешняя память была электромеханической. Сейчас начала применяться внешняя полупроводниковая память, получившая название **твёрдотельные накопители**

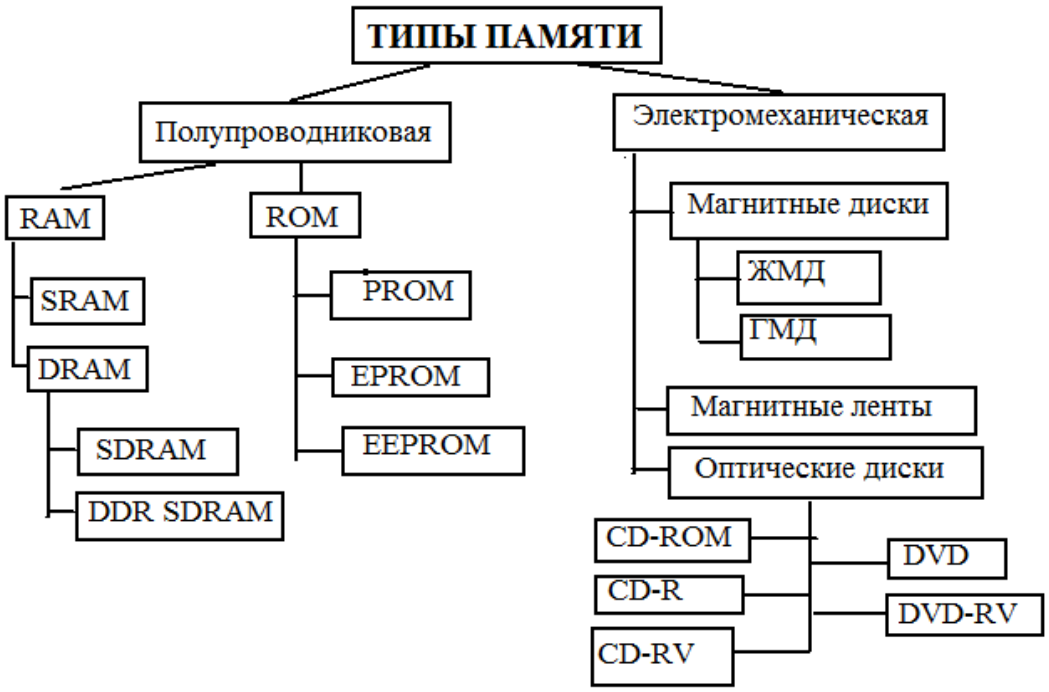


Рис. 10.2. Классификация памяти ЭВМ

Полупроводниковая память

RAM (Random Access Memory) - память с произвольным доступом - предназначена как для чтения так и для записи данных. Виды RAM:

- **SRAM** - статическая память, хранит информацию в течение всего времени, пока есть питание, быстрая, дорогая (6 транзисторов на 1 бит), применяется в кэш.
- **DRAM** - динамическая, асинхронная память. Хранит информацию в виде заряда на конденсаторе в течении примерно десяти мксек, требует периодического обновления (восстановления, регенерации) данных. Не синхронизирована с процессором. DRAM дешевле SRAM и применяется в ОЗУ.
- **SDRAM** отличается от DRAM тем, что синхронизирована с процессором
- **DDR SDRAM** - Double Data Rate SDRAM - с двойной скоростью, использует оба фронта тактового импульса.

ROM (Read Only Memory) - буквально - **память только для записи,** иначе называется постоянной памятью или **ПЗУ** (**постоянное запоминающее устройство**). Виды ROM:

- **ROM** - информация записывается при изготовлении на заводе,
- **PROM** - программируемая. В исходном состоянии вся память обнулена. Информацию можно записать с помощью программатора путём физического пережигания перемычек. Ненадёжная из-за

непредсказуемого распространения продуктов пережигания внутри устройства.

- **EPROM** (**Erasable Programmable Read Only Memory**) - стираемая, программируемая полупроводниковая память. Для записи нужно подать высокое напряжение, а для стирания нужно EPROM извлечь из компьютера и использовать ультафиолетовое облучение. EPROM следовало бы назвать полупостоянной памятью, так как запись в неё не невозможна, а затруднена, т. е. невозможна в оперативном режиме.
- **EEPROM** (**Electrically Erasable Programmable Read Only Memory**) - электрически стираемое перепрограммируемое ПЗУ (ЭСППЗУ). Память такого типа может стираться и заполняться данными до миллиона раз, поэтому входящее в её название **Read Only** вводит в заблуждение.

EEPROM отличается от RAM ограниченным количеством циклов записи, поэтому память этого типа не может использоваться в качестве оперативной. При выполнении многих программ может происходить запись в одну ячейку несколько миллионов раз.

Разновидностью EEPROM является **флеш-память**, из которой считывать ячейки можно по одной, а записывать блоками. Флеш-память используется во флешках, картах памяти и твёрдотельных накопителях.

Флешка - это жаргонная производная (сленг) от строгого технического термина **USB-флеш-накопитель**. Ёмкость флешек приближается к ёмкости винчестеров. Компактность, высокое быстродействие, удобство подключения через **USB-порт** сделали флешки самым популярным внешним накопителем.

Карта памяти - крошечная прямоугольная пластинка со срезанным углом и плоскими контактами, содержит в себе флеш память в виде интегральной схемы, применяется в мобильных устройствах.

Твёрдотельный накопитель, иначе называемый твёрдотельным диском, служит для замены жёсткого диска.

10.1 Электромеханическая память

10.1.1. Магнитные диски

Магнитные диски делятся на гибкие и жёсткие. Гибкие магнитные диски практически вытеснены из употребления флешками. Жёсткие магнитные диски (рис. 10.3) иначе называют ***винчестерами***. Носителем информации на винчестере служит тонкий алюминевый или стеклянный

диск (иначе называемый пластиной, тарелочкой, блином) с магнитным покрытием. Поверхность диска идеально ровная. Несколько дисков закрепляются на одной оси, вокруг которой они вращаются с огромной скоростью (от 4200 до 15000 об./мин.). Такую скорость вращения имеют некоторые гироскопы. Магнитная головка парит в воздушном потоке на расстоянии около десяти нанометров над поверхностью диска. (Толщина человеческого волоса (50 микрометров) в 5000 раз больше). Перед остановкой вращения диска блок головок перемещается за его пределы (паркуется).



Рис. 10.3. Жёсткий магнитный диск (винчестер)

Информация записывается на концентрические дорожки, разделённые на секторы (рис. 10.4). Совокупность дорожек одинакового диаметра на всех поверхностях всех пластин называется цилиндром. Для начала операции чтения или записи блок магнитных головок перемещается на заданную дорожку и ждет, пока под головкой не окажется нужный сектор. Время произвольного доступа - среднее время, за которое винчестер выполняет операцию подвода головки чтения/записи к нужному участку магнитного диска, зависит от скорости вращения. Диапазон этого параметра - от 2,5 до 16мс, часто в спецификациях указывают среднее время доступа порядка 10 мс. За это время процессор успеет выполнить несколько миллионов операций. Скорость чтения-записи после подвода головок соизмерима со скоростью работы ОЗУ. Поэтому обмен информацией между магнитным диском и ОЗУ всегда идёт большими блоками, а не по одному слову.

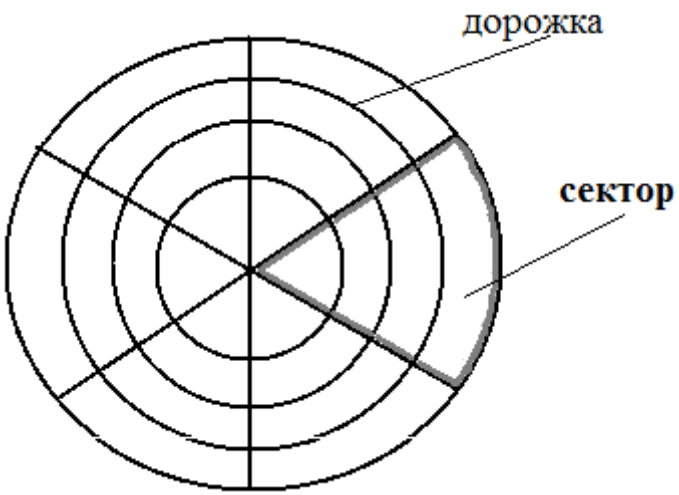


Рис. 10.4. Размещение информации на магнитном диске

10.1.2. Оптические диски

Оптические диски высокого качества могут хранить информацию, по заверению изготовителей, 100 лет. Это больше, чем на магнитной ленте (30 - 50 лет) и значительно больше, чем на магнитных дисках (3-10 лет) и флешках (3 - 7) лет.

Информация на оптических дисках записывается на одну спиралевидную дорожку, которая радиусом условно делится на части, называемые по аналогии с винчестером дорожками. Для считывания информации служит лазерный луч с длиной волны λ = 780 нм. Ширина впадины - 0.5 мк, длина впадины от 0.8 до 3 мк. Расстояние между дорожками - 1.6 мк. Длина спиральной дорожки около пяти километров. Ёмкость CD-ROM 650 - 800 Мбайт.

Рассмотрим устройство оптических дисков различных типов.

CD-ROM - компакт-диск, отливаемые на заводе. Его дорожка состоит из выступов и впадин (рис. 10.5), выдавливаемых в прозрачном слое поликарбонатного пластика, поверх которого наносится тонкий слой алюминия, а на него - защитное акриловое покрытие с этикеткой. Дорожка освещается лучом лазера. Глубина впадины равна четверти длины волны света. Свет, попадающий на выступ или впадину полностью отражается и попадает на детектор, а свет отражающийся от границы выступа и впадины взаимно поглощается со светом от источника. Здесь используется эффект "свет + свет = тьма". Отражающие участки кодируются нулём, а поглощающие - единицей.

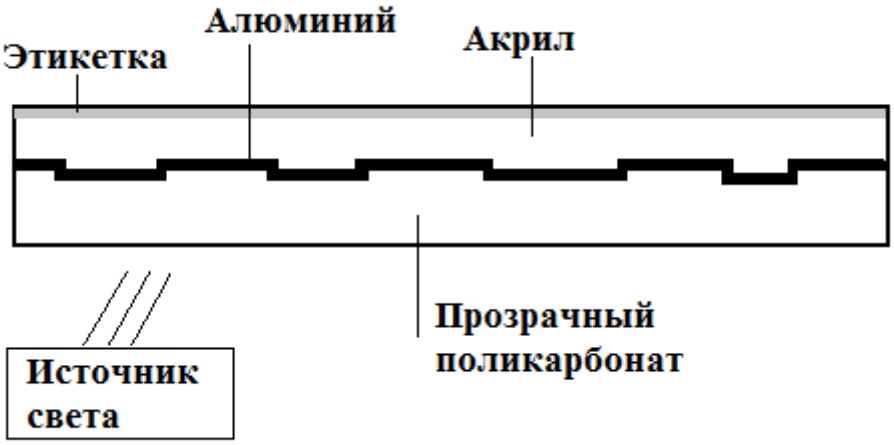


Рис. 10.5. Срез CD-ROM

Для литья CD-ROM предварительно изготавливается специальная матрица (форма) с выступами на месте впадин на будущих дисках и с впадинами на месте выступов. В эту матрицу заливается поликарбонат, в точности повторяющий рельеф матрицы. Такая технология возможна только при массовом производстве.

CD-R - оптический диск, спиралевидная дорожка на который наносится на заводе, а информация - на компьютере пользователя. CD-R покрыт специальным органическим веществом. Впадины на CD-R заменяются тёмными участками, создаваемыми путём нагрева этого вещества мощным лучом лазера. Записать информацию на CD-R можно только один раз.

CD-RW - оптический диск многократной записи - отличается от CD-R тем, что вместо органического слоя здесь используется специальный сплав серебра, индия, сурьмы и теллура. Если этот сплав нагреть до 500 ^{o}C, то после охлаждения он перейдёт в аморфное состояние и приобретёт способность поглощать свет. Но при повторном нагреве до 200 ^{o}C с последующим охлаждением он перейдёт в кристаллическое состояние и станет отражать свет. CD-RW может выдержать до 1000 перезаписей.

DVD

Для записи фильма ёмкости одного CD-ROM не хватает, поэтому, уменьшив все основные характеристики, получили диск **DVD** (**Digital Video Disc** — цифровой видеодиск), ёмкостью 4,7 Гбайт.

Технические характеристики DVD:

- длина волны лазера - 635 нм;
- минимальная длина впадины – 0.4 мк;
- расстояние между дорожками – 0.74 мк.

▪

10.1.3. Многоуровневая память

На рис. 10.6 изображена пятиуровневая структура памяти персональной ЭВМ. Зачем так сложно? Можно выделить три причины.

1. Стоимость памяти резко растёт с ростом её быстродействия. Чтобы цена ЭВМ была в доступных для покупателя пределах, приходится для хранения больших объёмов информации (сотни Гбайт) использовать самую дешёвую память. Сейчас - это магнитные диски и быстро дешевеющая флеш-память.
2. Самая быстрая память, построенная на элементах SRAM и DRAM, теряет информацию при выключении питания и в ней можно хранить программы и данные, необходимые только в текущем сеансе работы компьютера от момента его включения до момента выключения.
3. Обычно подряд выполняется целый блок последовательно расположенных в памяти команд. Данные также часто обрабатываются блоками, например, массивы. Устройства памяти могут передавать друг другу блоки данных без вмешательства процессора, работая параллельно с ним.

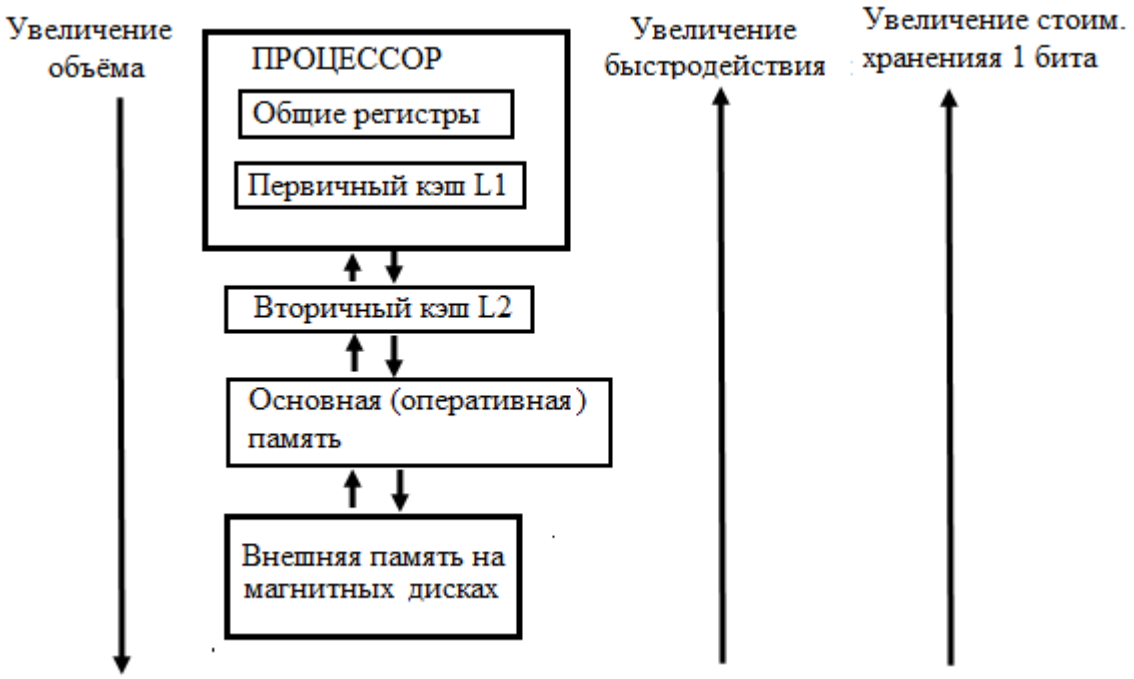


Рис. 10.6. Зависимость между стоимостью, быстродействием и объёмом памяти

Программа перед началом выполнения полностью или, если она большая, частично загружается из внешней памяти в оперативную. Часть

загруженной в оперативную память программы передаётся во вторичный кеш, а из вторичного кеша ещё меньшая часть програмрмы передаётся в первичный кеш. И только из первичного кеша загружается команда на регистр команд процессора для выполнения. Данные также распределяются блоками между устройствами памяти разных уровней.

После выполнения очередной команды процессор ищет следующую команду в первичном кеше. Если её там нет, то поиск переходит во вторичный кеш, оперативную память и, если программа была загружена в оперативную память не полностью, то и во внешнюю память. Поиск прекращается на том уровне, где была обнаружена искомая команда. Далее команда вместе с блоком, в котором она находится передаётся на следующие уровни, пока не попадёт в процессор.

Обмен между внешней и оперативной памятью управляется программным путём, а обмен с кешами - аппаратно. Первичный кеш ассоциативный и делится на две части. В одной хранится программа, в другой - данные. Здесь произошло отступление от принципа фон Неймана.

Лекция 11.

11. Внешние устройства

Ранее были рассмотрены относящиеся к устройствам ввода-вывода магнитные диски, ленты, оптические диски и кеш-память.

11.1 Устройства ввода

11.1.1 Клавиатура

Для программиста клавиатура является основным средством ввода информации в ЭВМ. Размер клавиатуры для настольных ПЭВМ оптимальный для человеческих рук. Набор и расположение клавиш соответствует принятому для PC IBM/IT. К сожалению, в ноутбуках клавиатуры отличаются большим разнообразием, в основном, в расположении навигационных клавиш.

Во многих редакторах текстов (Notepad++, UltraEdit), предназначенных для разработки программ, используются следующие сочетания клавиш, значительно ускоряющие редактирование текстов программ:

- Ctrl+A - выделить весь документ;
- Ctrl+S — сохранить файл;
- Ctrl+Z — отменить последнее изменение;
- Ctrl+F — открыть окно поиска;
- Ctrl+G — открыть окно для перехода к строке с заданным номером;
- Ctrl+H — открыть окно замены;
- Ctrl+C — копировать выделенный фрагмент;
- Ctrl+V — вставить выделенный фрагмент;
- Ctrl+X — вырезать выделенный фрагмент;

Все перечисленные клавиши находятся в левой нижней части клавиатуры (рис. 11.1).



Рис. 11.1. Часто используемые сочетания клавиш

Для быстрого перемещения по редактируемому тексту полезны следующие навигационные клавиши:

- Ctrl+Home — переместиться в начало документа;
- Ctrl+End — переместиться в конец документа;
- Home — переместиться в начало строки;
- End — переместиться в конец строки;

Клавиатуры по своей конструкции делятся на четыре вида:

- механические;
- мембранные;
- полумеханические;
- сенсорные.

В механических клавиатурах кнопка после нажатия возвращается пружиной. Достоинствами таких устройств являются долговечность и надежность, а также отсутствие эффекта «усталости» конструкции. Механические клавиши выдерживают гораздо большее количество нажатий, чем мембранные и полумеханические. Недостаток моделей данного типа – достаточно шумная работа. Кроме того, эти устройства довольно дорого стоят и их механизм не защищен от внешних воздействий. Правда, есть герметизированные модели, но они стоят еще дороже.

В мембранных клавиатурах при нажатии клавиши смыкаются две мембраны (обычно они выглядят как диски на пластиковой пленке, поэтому и употребляется наименование «пленочная клавиатура»). За возврат клавиши отвечает эластичный купол (с углублением в центре). Для разделения мембран служит промежуточная пленка с отверстиями. Благодаря тому что контакты находятся на внутренних сторонах пленок, эта конструкция хорошо защищена от пыли и влаги. В максимально герметизированной реализации мембранная клавиатура содержит внутри единый резиновый «коврик» с расположенными под клавишами выступающими куполами. К плюсам мембранных клавиатур можно отнести тихую работу, «мягкость» (легкость нажатия на клавиши), упомянутую выше защищенность от внешних воздействий и низкую цену. Единственным их недостатком является меньшая по сравнению с другими типами конструкций долговечность. Со временем упругие элементы теряют свою эластичность и требуется (иногда очень) сильное нажатие для срабатывания клавиши.

Описанная мембранная конструкция не единственная, существует множество других, отличающихся количеством, формой, материалом упругих элементов.

В полумеханических клавиатурах используются более долговечные металлические контакты (в дорогих моделях они могут быть даже позолоченными), распаянные на печатной плате. В остальном они не отличаются от мембранных: в исходное состояние клавиши в них также возвращаются с помощью эластичного пластикового купола. Только стоят такие устройства дороже.

Сенсорная клавиатура имеет тот же принцип действия, что и сенсорные панели, описанные ниже в этой лекции. Её можно представить как сенсорный экран или сенсорную панель, с нарисованными клавишами. Сенсорные клавиатуры дорогие, непривычны, широкого распространения не получили.

11.1.2. Компьютерные мыши

Компьютерная мышь служит для управления движением курсора по экрану монитора. Человек передвигает мышку по плоской поверхности, а курсор повторяет эти движения в соответствующем масштабе на экране монитора. Мышки бывают двух типов: механические с шаровым приводом и оптические.

У **мышки с шаровым приводом** на дне имеется выступающий из корпуса обрезиненный стальной шарик. Его вес и резиновое покрытие обеспечивают хорошее сцепление с поверхностью, по которой движется мышка. При перемещении мышки шарик вращается. Два прижатых к шарику взаимно перпендикулярных ролика снимают его движения по каждому из измерений и передают их на датчики угла поворота, преобразующие эти движения в электрические сигналы.

Основным недостатком механической мышки является прилипание к роликам грязи собираемой со стола шариком. Грязь легко удалить, после чего мышка снова работает нормально. (Грязь состоит из смеси жира выделяемого кожей ладони и пыли, оседающей на стол.)

Идея **оптической мыши** несколько фантастична. В нижней части мыши установлена специальная быстрая видеокамера. Она непрерывно делает снимки поверхности стола и, сравнивая их, определяет направление и величину смещения мышки. Специальная контрастная подсветка поверхности светодиодом или лазером облегчает работу камеры. Появление оптической мыши стало возможным благодаря встроенному в неё мощному микропроцессору, обрабатывающему "карту стола". Теоретически возможно обмануть оптическую мышь, если передвигать её по абсолютно ровной одноцветной поверхности.

Оптические мышки также страдают от пыли, но в гораздо меньшей степени, чем механические, и почти вытеснили последние.

Есть ещё один вид мышек - **гироскопические**, но они используются очень редко. Такая мышка, благодаря встроенному в неё гироскопу передаёт своё положение в пространстве.

11.1.3. Трекбол и джойстик

Трекбол (англ. trackball) - ручное указательное устройство ввода информации для компьютера (рис. 11.2). Аналогичен мыши по принципу действия и по функциям: позволяет вводить информацию об относительном перемещении путём вращения рукой закреплённого в корпусе шара и подавать команды нажатиями на кнопки.



Рис. 11.2 Трекбол.

Джо́йстик (англ. joystick, дословно «палочка радости») - устройство ввода информации в компьютер, которое представляет собой качающуюся в двух плоскостях вертикальную ручку (рис.11.3).

Джойстик позволяет управлять виртуальным объектом в двух- или трехмерном пространстве. Помимо координатных осей «X» и «Y», некоторые джойстики способны предоставлять координаты оси «Z», посредством вращения ручки джойстика вокруг её оси, либо с помощью дополнительного управляющего элемента на основании джойстика. Программное обеспечение, получив информацию о координатах «X-Y-Z», позволяет пользователю управлять неким виртуальным объектом, отображаемым на мониторе. На ручке джойстика и на его основании обычно располагаются кнопки, переключатели, слайдеры, крестовина и другие управляющие элементы различного назначения.

Широкое применение джойстик получил в компьютерных играх, мобильных телефонах. В русском языке словом «джойстик» также иногда называют другое устройство ввода в консольных видеоиграх **геймпад.**



Рис. 11.3 Джойстик.

11.1.4. Сенсорная панель

Се́нсорная пане́ль, (тачпа́д) - указательное (координатное) устройство ввода, предназначенное для управления курсором и отдачи различных команд компьютеру, телефону или другому электронному оборудованию.

Принцип работы сенсорной панели заключается в том, что при прикосновении к какой-либо точке панели пальцем или стилусом, меняются электрические характеристики этой точки и можно определить её координаты. (Стилус — небольшая металлическая или пластиковая палочка со специальным силиконовым наконечником.) Прозрачную сенсорную панель объединяют с экраном монитора. Такой экран называется **сенсорным экраном** и служит для ввода-вывода.

Сенсорную панель часто называют сенсорным экраном, что приводит к изрядной путанице.

По своему устройству сенсорные панели делятся на несколько видов.

Резистивная сенсорная панель (рис. 11.4) состоит из стеклянной панели и гибкой пластиковой мембраны. И на панель, и на мембрану нанесено резистивное покрытие. Пространство между стеклом и мембраной заполнено микроизоляторами, которые равномерно распределены по активной области экрана и надёжно изолируют проводящие поверхности. Когда на экран нажимают, панель и мембрана замыкаются, и контроллер с помощью аналогово-цифрового преобразователя регистрирует изменение сопротивления и преобразует его в координаты прикосновения (X и Y). Считывание происходит в два этапа:

1. Между верхними электродами подаётся напряжение +5 В, нижние заземляются. Между точкой нажатия и землёй измеряется напряжение. Это напряжение соответствует Y-координате экрана. Этот этап показан на рис. 11.4.

2. Аналогично между нижними электродами подаётся +5 В, а верхние заземляются. В точке нажатия считывается X-координата.

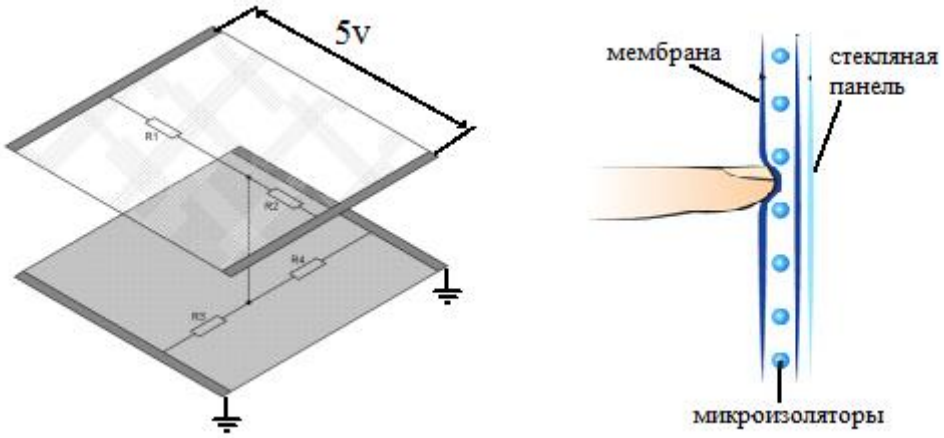


Рис. 11.4. Устройство 4-проводной резистивной сенсорной панели.

Матричная сенсорная панель аналогична резистивной, но упрощена до предела. На стекло нанесены горизонтальные проводники, на мембрану — вертикальные. При прикосновении к экрану проводники соприкасаются. Контроллер определяет, какие проводники замкнулись, и передаёт в микропроцессор соответствующие координаты.

Матричные сенсорные панели имеют очень низкую точность. Элементы интерфейса приходится специально располагать с учётом клеток матричного экрана. Единственное достоинство — простота, дешевизна и неприхотливость. Обычно матричные экраны опрашиваются по строкам (аналогично матрице кнопок).

В **ёмкостной сенсорной панели** (или поверхностно-ёмкостной) используется тело человека в качестве конденсатора, проводящего переменный ток (рис. 11.5).

Ёмкостная сенсорная панель представляет собой стеклянную панель, покрытую прозрачным резистивным материалом. Через электроды, расположенные по углам панели, подаётся на проводящий слой небольшое переменное напряжение (одинаковое для всех углов). При касании панели пальцем или другим проводящим предметом появляется утечка тока. При этом чем ближе палец к электроду, тем меньше сопротивление панели, а значит, сила тока больше. Ток во всех четырёх углах регистрируется датчиками и передаётся в контроллер, вычисляющий координаты точки касания.

Ёмкостные сенсорные панели надёжны, выдерживают порядка 200 млн нажатий, не пропускают жидкости и отлично терпят токонепроводящие загрязнения. Прозрачность на уровне 90%. Недостатком является не очень прочное проводящее покрытие, расположенное прямо на внешней

поверхности. Поэтому ёмкостные панели широко применяются в автоматах, лишь установленных в защищённом от непогоды помещении. Не реагируют на руку в перчатке, точнее, на прикосновение любым диэлектриком.

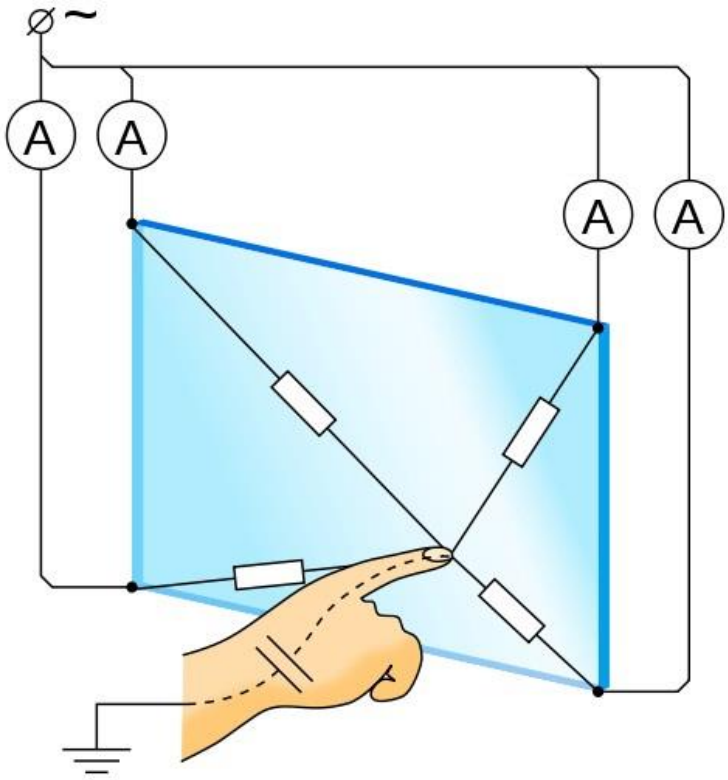


Рис. 11.5. Устройство емкостной сенсорной панели.

Проекционно-емкостная сенсорная панель имеет более сложное устройство по сравнению с поверхностно-ёмкостной. На внутренней стороне экрана нанесена сетка электродов (рис. 11.6). Электрод вместе с телом человека образует конденсатор; электроника измеряет ёмкость этого конденсатора (подаёт импульс тока и измеряет напряжение).

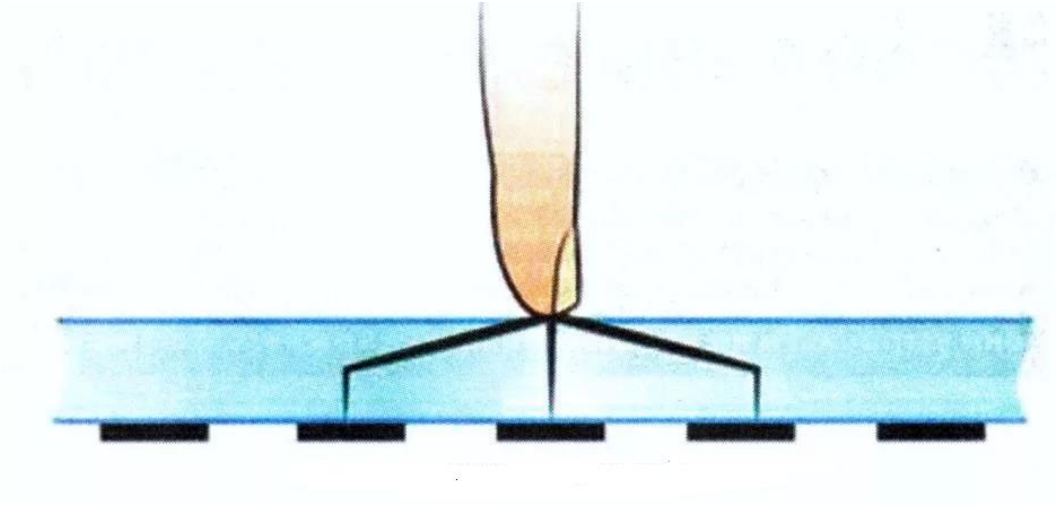


Рис. 11.6. Устройство проекционно-емкостной сенсорной панели.

Прозрачность таких экранов до 90%, температурный диапазон чрезвычайно широк. Очень долговечны (узкое место — сложная электроника, обрабатывающая нажатия). На проекционно-ёмкостных экранах может применяться стекло толщиной вплоть до 18 мм, что обеспечивает большую «вандалоустойчивость». На непроводящие загрязнения не реагируют, проводящие легко подавляются программными методами. Поэтому проекционно-ёмкостные сенсорные панели широко применяются и в персональной электронике, и в автоматах, в том числе установленных на улице. Многие разновидности поддерживают мультитач.

Мультитач (от англ. Multi-touch — «множественное касание») — функция сенсорных систем ввода (сенсорный экран, сенсорная панель), осуществляющая одновременное определение координат двух и более точек касания.

Другие типы сенсорных панелей используют различные физические принципы: пьезоэффект, оптические эффекты, электромагнитную индукцию.

11.5 Сканеры

Сканер (от англ. scan - «пристально разглядывать, рассматривать») - устройство ввода, которое, анализируя какой-либо объект (обычно изображение, текст), создаёт его цифровое изображение. Процесс получения этой копии называется сканированием.

Сканеры служат для копирования в ЭВМ изображения с бумажного носителя. Можно скопировать изображение любого объекта, который поместится на рабочее поле сканера, например, кисти руки.

Бывают ручные, рулонные, планшетные и проекционные сканеры. Разновидностью проекционных сканеров являются слайд-сканеры, предназначенные для сканирования фотоплёнок. В высококачественной полиграфии применяются барабанные сканеры, в которых в качестве светочувствительного элемента используется фотоэлектронный умножитель (ФЭУ).

Принцип работы однопроходного планшетного сканера состоит в том, что вдоль сканируемого изображения, расположенного на прозрачном неподвижном стекле, движется сканирующая каретка с источником света. Отражённый свет через оптическую систему сканера (состоящую из объектива и зеркал или призмы) попадает на 3 расположенных параллельно друг другу фоточувствительных полупроводниковых

элемента на основе ПЗС (прибор с зарядовой связью), каждый из которых принимает информацию о компонентах изображения.

11.2 Устройства вывода

11.2.1 Мониторы и дисплеи

Дисплей (англ. display — показывать) - синоним экрана, на который выводится изображение. Сенсорный дисплей - дисплей с прикреплённой к нему сенсорной панелью. Монитор состоит из дисплея, устройства управления, блока питания и корпуса. Наиболее распространённые типы дисплеев: на электронно-лучевой трубке (ЭЛТ), жидкокристаллический, плазменнй. Применяются также светодиодные и лазерно-фосфорные дисплеи.

ЭЛТ-монитор представляет собой прибор, в котором пучки электронных лучей вырисовывают изображение на внутренней стороне электронно-лучевой трубки (рис.11.7).

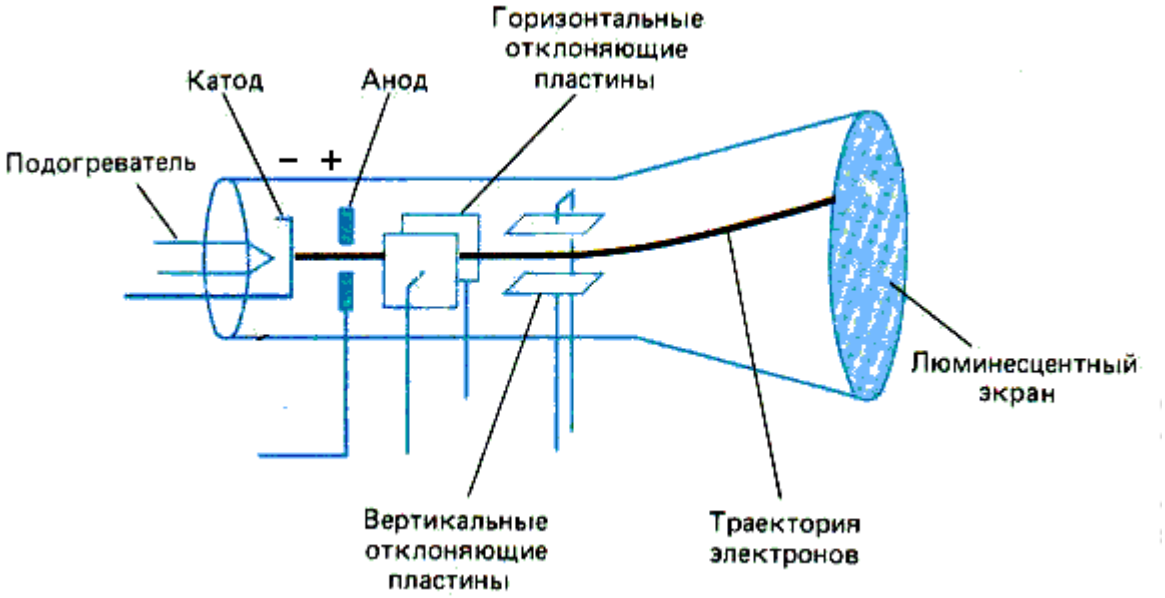


Рис. 11.7. Электронно-лучевая трубка

В ЭЛТ используется свойство нагретых металлов испускать электроны. Между нагретым катодом и анодом подаётся высокое (несколько тысяч или десятков тысяч вольт) напряжение. Электроны, будучи отрицательно заряженными частицами, устремляются к положительному аноду, проскакивают сквозь него, попадают в электромагнитное поле отклоняющей системы и ударяются в люминесцентный экран. Происходит вспышка света.

Люминесцентное покрытие экранов состоит из рядов пикселей. Количество пикселей по горизонтали и по вертикали определяет физическую разрешающую способность экрана. Пиксель цветного экрана

состоит из трёх элементов: красного, зелёного и синего, катод излучает три пучка электронов, которые попадают в соответствующие элементы пикселя. Траекторией и интенсивностью пучков электронов управляет сложная электронная система. Вспышка света при попадании электрона в люминофор кратковременная, поэтому электронный луч, пробегая по всем пикселям экрана, непрерывно восстанавливает изображение. Частота восстановления изображения на экране обычно 50-75 Гц. При такой частоте, человек не видит мелькания изображения, но глаз чувствует и утомляется. Чем больше частота восстановления, тем меньше утомляются глаза. Часть электронов вылетает из экрана и вблизи экрана существует слабое бета-излучение. Поэтому не следует слишком близко приближать глаза к экрану.

ЭЛТ-мониторы позволяют формировать очень качественное изображение, но главный их недостаток - большие габариты - стал причиной их вытеснения мониторами других типов, главным образом, жидкокристаллическими.

Жидкокристаллические (ЖК-) дисплеи получили широчайшее распространение благодаря тому, что они плоские. Это свойство делает возможным существование портативных устройств: ноутбуков, планшетов, мобильных телефонов. ЖК-дисплеи потребляют в среднем меньше энергии, чем ЭЛТ-дисплеи, не имеют мерцаний и вредных излучений. Главный недостаток ЖК-дисплеев - искажение цветов при разных углах зрения. Принцип действия ЖК-дисплеев заключается в возможности поворота плоскости поляризации луча света. На рис.11.8 неполяризованный луч света проходя через прозрачную пластину с вертикальными штрихами, поляризуется в вертикальной плоскости. Если плоскость поляризации не изменить, то этот луч не пройдёт через пластину с горизонтальными штрихами. На рисунке плоскость поляризации поворачивается на 90^{o} и вторая пластина становится полностью прозрачной для луча. Если луч развернуть на меньший угол, то вторая пластина пропустит меньше света. Для разворота плоскости поляризации луча света и служат управляемые с помощь электромагнитного поля жидкие кристаллы.

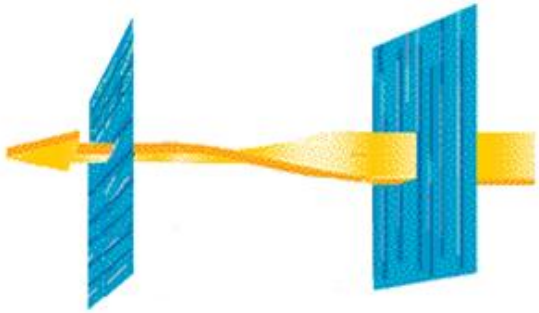


Рис. 11.8. Поляризация света

Жидкие кристаллы имеют структуру вязкой жидкости, которая состоит из молекул дискообразной формы. Ориентация этих молекул может изменяться при взаимодействии с электромагнитными полями. Изображение на ЖК-дисплее строится из пикселей. Каждый пиксель состоит из трёх субпикселей: красного (рис. 11.9), зелёного и синего. Субпиксель состоит из

- вертикального и горизонтального поляризаторов (пластин с вертикальными и горизонтальными штрихами);
- двух прозрачных электродов, служащих для управления положением жидких кристаллов;
- жидких кристаллов;
- цветового фильтра.

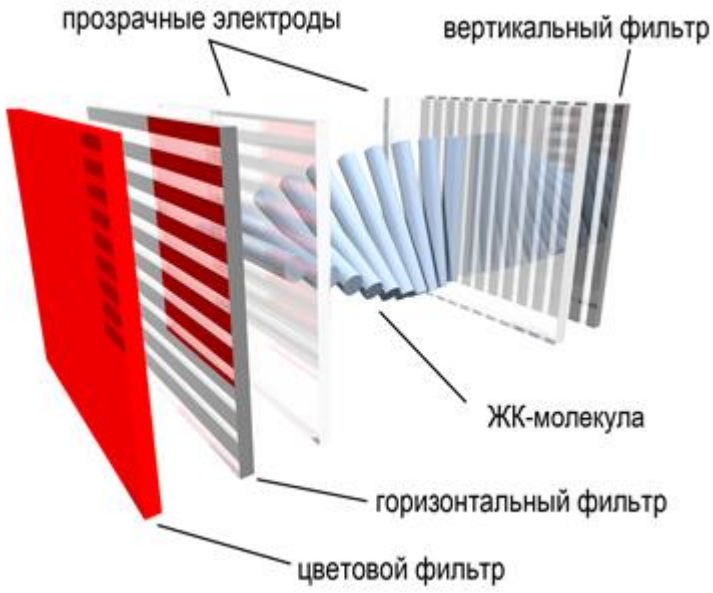


Рис. 11.9. Субпиксель цветного ЖК-дисплея

Жидкие кристаллы под действием переменного напряжения, подаваемого на электроды, меняют своё положение в пространстве. Плоскость поляризации луча света, при его прохождении через жидкие кристаллы меняет свой угол относительно оси луча в соответствии с

положением молекул жидкого кристалла. На рис.10.7 кристаллы развернулись так, что угол между плоскостью поляризации луча с его осью изменился на 90° и луч свободно прошёл через горизонтальный поляризатор.

Для управления положением субпикселей используются пассивные и активные матрицы. ЖК-дисплей на пассивной матрице использует две системы прозрачных электродов: вертикальных и горизонтальных. В ЖК-дисплей на активной матрице используется для управления каждым пикселем тонкоплёночный транзистор.

Плазменные дисплеи используют явление свечения люминофора при попадании на него ультрафиолетовых лучей. Лучи возникают при электрическом разряде в ионизированном газе (плазме). Каждый субпиксель представляет собой миниатюрную люминисцентную лампу.

11.2.2 Принтеры

Матричные принтеры по принципу действия напоминают пишущие машинки. Между бумагой и пишущим узлом (печатающей головкой) находится красящая лента. При ударе элементов пишущего узла по ленте на бумаге остаётся отпечаток. Элементами печатающей головки матричного принтера является вертикальный ряд крохотных стержней (иголок), управляемых электромагнитами. Печатаемое изображение (символ) получается в виде комбинации точек, оставляемых стержнями. Для улучшения качества печати стержни делают квадратного сечения. Матричные принтеры самые дешёвые в эксплуатации и, не смотря на невысокое качество печати и шумность, применяются там где нужно печатать документы "под копирку" и к качеству печати не предъявляется высоких требований.

Струйные принтеры, так же как и матричные, печатают изображение, состоящее из точек, но для печати точки используется чернильница с узким отверстием (фильерой), через которую капелька чернил под давлением выстреливается на бумагу. На струйных принтерах при использовании специальной бумаги и чернил получаются цветные изображения наилучшего качества, передающие мельчайшие оттенки цвета. Недостаток струйных принтеров - очень высокая стоимость чернильных картриджей.

Лазерные принтеры имеют более сложное устройство, чем матричные и струйные, но и высокое качество и самую высокую скорость печати. Основной узел лазерного принтера - фотобарабан - цилиндр с покрытием из фотополупроводника (материала, способного менять своё электрическое

сопротивление при освещении). Перед формированием изображения вся поверхность фотобарабана заряжается отрицательно, затем с тех мест, в которых должно быть изображение, лучом лазера снимается часть заряда, и наконец, на вал наносится красящий порошок (тонер). Тонер прилипает там где частично или полностью снят электрический заряд. Следующий этап - перенесение тонера с фотобарабана (получение оттиска) на бумагу. Последний этап - закрепление тонера на бумаге путём её нагрева. Это самый неприятный этап, так как при нагреве выделяется озон (и не только). Лазерные принтеры широко применяются там, где нужно много печатать. Из-за выделения озона помещение с лазерным принтером должно хорошо проветриваться.

Плоттеры можно назвать промышленными принтерами. Их отличительная особенность - возможность печатать изображения большого формата. Второе название плоттеров - графопостроители. Различают два типа плоттеров - рулонные и планшетные. Различие очевидно из названий.



Рис. 11.10. Планшетный плоттер

Планшетный плоттер (рис 11.10) позволяет не только печатать но и вырезать из материала детали заданной формы, если печатающий узел заменить режущим инструментом (например, лазерным).