МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО
ОБРАЗОВАНИЯ «САМАРСКИЙ ГОСУДАРСТВЕННЫЙ
АЭРОКОСМИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П.
КОРОЛЕВА (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
(СГАУ)

Организация ЭВМ и вычислительных систем

Материалы электронного контента по дисциплине

Работа выполнена по мероприятию блока 1 «Совершенствование образовательной деятельности» Программы развития СГАУ на 2009 – 2018 годы по проекту «Разработка образовательного контента в рамках мастер-класса по внедрению и использованию СЭДО в реальном учебном процессе» Соглашение № 1/27 от 03 июня 2013 г.

САМАРА, 2013

УДК 004.2 (075)
ББК 32.9я7 З
О-64

Автор-составитель**: Зеленко Лариса Сергеевна**

Рецензент: Симонова Е.В., к.т.н, доцент кафедры ИСТ СГАУ

Организация ЭВМ и вычислительных систем [электронный ресурс]: материалы электронного контента по дисциплине/ М-во образования и науки РФ, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т); - авт.-сост. *Л.С. Зеленко*. - Электрон. текстовые и граф. дан. (2,26 Мбайт). - Самара, 2013. - 1 эл. опт. диск (CD-ROM).

Режим доступа: http://do-ps.ssau.ru/course/category.php?id=7.

В состав электронного контента входят следующие материалы:

1.Курс лекций по Организации ЭВМ и ВС.

2.Методические указания к лабораторному практикуму на Ассемблере.

3.Список вопросов к зачету по организации ЭВМ и ВС.

4.Тесты для итогового контроля знаний.

В курсе лекций изложены теоретические сведения, необходимые для понимания внутренней организации ЭВМ: приведены ее характеристики, классификации по разным критериям, а также изложены основные принципы построения современных ЭВМ и вычислительных систем. Описывается классификация элементов и узлов ЭВМ и дается подробное описание их характеристик и возможностей. В методических указаниях изложены основные теоретические сведения по программированию на языке Ассемблера, приведены решения типовых задач, а также включены варианты заданий по темам лабораторного практикума.

Материалы электронного контента предназначены на студентов факультета информатики, обучающихся по бакалаврской программе направления 010300.62 «Фундаментальная информатика и информационные технологии» (ФГОС-3), которые изучают дисциплину «Организация ЭВМ и вычислительных систем» во 2 семестре.

Материалы разработаны на кафедре программных систем.

Самарский государственный

аэрокосмический университет, 2013

ОРГАНИЗАЦИЯ ЭВМ И ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Курс лекций

Автор: доцент кафедры
программных систем,
к.т.н. Зеленко Л.С.

Самара 2013

СОДЕРЖАНИЕ

ГЛАВА 1 ПРИНЦИПЫ ПОСТРОЕНИЯ И АРХИТЕКТУРА ЭВМ

1.1. Основные характеристики ЭВМ

Первые электронные вычислительные машины (ЭВМ) появились всего лишь в середине прошлого века. За это время микроэлектроника, вычислительная техника и вся индустрия информатики стали одними из основных составляющих мирового научно-технического прогресса. Влияние вычислительной техники на все сферы деятельности человека продолжает расширяться вширь и вглубь. В настоящее время ЭВМ используются не только для выполнения сложных расчетов, но и в управлении производственными процессами, в образовании, здравоохранении, экологии и т.д. Это объясняется тем, что ЭВМ способны обрабатывать любые *виды информации*:

- числовую,
- текстовую,
- табличную,
- графическую
- видео,
- звуковую.

> ***Электронная вычислительная машина (ЭВМ)*** - комплекс технических и программных средств, предназначенный для автоматизации подготовки и решения задач пользователей.

Под ***пользователем*** понимают человека, в интересах которого проводится обработка данных на ЭВМ. В качестве пользователя могут выступать заказчики вычислительных работ, программисты, операторы. ♦

Требования пользователей к выполнению вычислительных работ удовлетворяются специальным подбором и настройкой технических и программных средств. Обычно эти средства взаимосвязаны и объединяются в одну структуру.

> ***Структура*** - совокупность элементов и их связей, которая рассматривается в конкретном приложении.

Различают структуры *технических*, *программных* и *аппаратно-программных* средств. Выбирая ЭВМ для решения своих задач, пользователь интересуется функциональными возможностями технических и программных модулей (как быстро может быть решена задача, насколько ЭВМ подходит для решения данного круга задач,

☞ Как правило, время подготовки задач во много раз превышает время их решения.

какой сервис программ имеется в ЭВМ, возможности диалогового режима, стоимость подготовки и решения задач и т.д.). При этом пользователь интересуется не конкретной технической и программной реализацией отдельных модулей, а более общими вопросами возможности организации вычислений. Последнее включается в понятие архитектуры ЭВМ, содержание которого достаточно обширно.

Архитектура ЭВМ **–** это описание цифровой вычислительной системы на некотором общем уровне, включающем

- пользовательских возможностей программирования;
- Описание системы команд;
- Средства пользовательского интерфейса;
- Организацию памяти и систему адресации;
- Операции ввода/вывода и управления и т.п.

В контексте разработки вычислительной системы (ВС) и проектирования термин *«Архитектура ЭВМ»* используется для описания принципов действия, конфигурации и взаимного соединения логических узлов ЭВМ. ***Архитектура ЭВМ*** - это многоуровневая иерархия аппаратно-программных средств, каждый из уровней допускает многовариантное построение и применение. Конкретная реализация уровней определяет особенности структурного построения ЭВМ.

Детализацией архитектурного и структурного построения ЭВМ занимаются различные категории специалистов вычислительной техники. *Инженеры-схемотехники* проектируют отдельные технические устройства и разрабатывают методы их сопряжения друг с другом. *Системные программист* создают программы управления техническими средствами, информационного взаимодействия между уровнями, организации вычислительного процесса. *Программисты-прикладники* разрабатывают пакеты программ более высокого уровня, которые обеспечивают взаимодействие пользователей с ЭВМ и необходимый сервис при решении ими своих задач.

Самого же пользователя интересуют обычно более общие вопросы, касающиеся его взаимодействия с ЭВМ (человеко-машинного интерфейса), начиная со следующих групп характеристик ЭВМ, определяющих ее структуру (см. рисунок 1.1):

- ***технические и эксплуатационные характеристики ЭВМ*** (быстродействие и производительность, показатели надежности, достоверности, точности, емкость оперативной и внешней памяти, габаритные размеры,

стоимость технических и программных средств, особенности эксплуатации и др.);

- ***характеристики и состав функциональных модулей базовой конфигурации ЭВМ***; возможность расширения состава технических и программных средств; возможность изменения структуры;
- ***состав программного обеспечения ЭВМ и сервисных услуг*** (операционная система или среда, пакеты прикладных программ, средства автоматизации программирования).

Рисунок 1.1 – Основные характеристики ЭВМ

Одной из важнейших характеристик ЭВМ является ее ***быстродействие***, которое характеризуется числом команд, выполняемых ЭВМ за одну секунду. Поскольку в состав команд ЭВМ включаются операции, различные по длительности выполнения и по вероятности их использования, то имеет смысл характеризовать его или средним быстродействием ЭВМ, или предельным (для самых «коротких» операций типа «регистр-регистр»). Современные вычислительные машины имеют очень высокие характеристики по быстродействию, измеряемые десятками и сотнями миллионов операций в секунду. Например, в ближайшее время ожидается появление микропроцессора совместного производства фирм Intel и Hewlett-Packard (шифр P7), быстродействие которого должно достичь миллиарда операций в секунду.

Реальное или эффективное быстродействие, обеспечиваемое ЭВМ, значительно ниже, и оно может сильно отличаться в зависимости от класса решаемых задач. Сравнение по быстродействию различных типов ЭВМ, резко отличающихся друг от друга своими характеристиками, не обеспечивает достоверных оценок. Поэтому очень часто вместо характеристики быстродействия используют связанную с ней характеристику ***производительности*** (эффективность) – объем работ, осуществляемых ЭВМ в единицу времени. Например, можно определять этот параметр числом задач, выполняемых за определенное время.

$P = \frac{K_з}{t}$, где $К_з$ – количество выполненных задач за промежуток времени длиной t.

Однако сравнение по данной характеристике ЭВМ различных типов может вызвать затруднения. Поскольку оценка производительности различных ЭВМ является важной практической задачей, хотя такая постановка вопроса также не вполне корректна, были предложены к использованию *относительные характеристики производительности*. Так, например, фирма Intel для оценки процессоров предложила тест, получивший название индекс iCOMP (Intel Comparative Microprocessor Performance). При его определении учитываются четыре главных аспекта производительности: работа с целыми числами, с плавающей точкой, графикой и видео. Данные имеют 16- и 32-разрядное представление. Каждый из восьми параметров при вычислении участвует со своим весовым коэффициентом, определяемым по усредненному соотношению между этими операциями в реальных задачах (таблица 1.1).

Таблица 1.1 - Индекс iCOMP

Типы данных	Тест	Весовой коэффициент в iCOMP, %
16- разрядные целые	PC Labs v7.01; Processor	52
32- разрядные целые	SPECint92	15
16- разрядные, графика	PC Labs WinBench v3.11	10
32- разрядные, графика	SPECint92	5
16- разрядные, видео	PC Labs v7.01*; Video*	5
32- разрядные, видео	SPECint92	5
16-разрядные вещественные	Power Meter vl.7; Whetstone PC Labs v7.01; Math Coprocessor	21
32-разрядные вещественные	SPECfp92	5

По индексу iCOMP микропроцессор Pentium 100 имеет значение 810, а Pentium 133 - 1000.

Другой важнейшей характеристикой ЭВМ является ***емкость запоминающих устройств,*** которая измеряется количеством структурных единиц информации, которое может одновременно находиться в памяти. Этот показатель позволяет определить, какой набор программ и данных может быть одновременно размещен в памяти.

Наименьшей структурной единицей информации является ***бит*** - одна двоичная цифра. Как правило, емкость памяти оценивается в более крупных единицах измерения - ***байтах***. Современные единицы измерения приведены в таблице 1.2.

Таблица 1.2 – Единицы измерения информации

1 байт = 8 бит
1 слово = 2 байта = 16 бит
1 Кбайт = 2^{10} (1024) байта

1 Мбайт = 2^{10}Кбайта = = 2^{20} байта
1 Гбайт = 2^{10} Мбайта = 2^{20} Кбайта = 2^{30} байта
1 Тбайт = 2^{10} Гбайта = 2^{20} Мбайта = 2^{30} Кбайт = 2^{40} байта

Обычно отдельно характеризуют емкость оперативной памяти и емкость внешней памяти. В настоящее время персональные ЭВМ могут иметь емкость оперативной памяти, равную 1-16 Гбайтам и даже больше. Этот показатель очень важен для определения, какие программные пакеты и их приложения могут одновременно обрабатываться в машине.

Емкость внешней памяти зависит от типа носителя. Так, емкость одной дискеты составляет 1,44 Мбайта в зависимости от типа дисковода и характеристик дискет. Емкость жесткого диска может достигать нескольких десятков и даже сотен Гбайтов, емкость компакт-диска (CD ROM) - сотни Мбайтов (640 Мбайт и выше) и т.д. Емкость внешней памяти характеризует объем программного обеспечения и отдельных программных продуктов, которые могут устанавливаться в ЭВМ. Например, для установки операционной среды Windows 95 требуется объем памяти жесткого диска более 100 Мбайт и не менее 8-16 Мбайт оперативной памяти ЭВМ.

Надежность - это способность ЭВМ при определенных условиях выполнять требуемые функции в течение заданного периода времени (стандарт ISO 23 82/14-78)♦.

Высокая надежность ЭВМ закладывается в процессе ее производства. Переход на новую элементную базу - сверхбольшие интегральные схемы (СБИС) резко сокращает число используемых интегральных схем, а значит, и число их соединений друг с другом. Хорошо продуманы компоновка компьютера и обеспечение требуемых режимов работы (охлаждение, защита от пыли). Модульный принцип построения позволяет легко проверять и контролировать работу всех устройств, проводить диагностику и устранение неисправностей.

Точность - возможность различать почти равные значения (стандарт ISO 2382/2-76).

Точность получения результатов обработки в основном определяется разрядностью ЭВМ, а также используемыми структурными единицами представления информации (байтом, словом, двойным словом).

Во многих применениях ЭВМ не требуется большой точности, например, при обрабатывании текстов и документов, при управлении технологическими процессами. В этом случае достаточно использовать 8-и, 16- разрядные двоичные коды.

♦ ISO (International Standard Organization) – Международная организация стандартов

При выполнении сложных расчетов требуется использовать более высокую разрядность (32, 64 и даже более). Поэтому все современные ЭВМ имеют возможность работы с 16- и 32- разрядными машинными словами. С помощью средств программирования языков высокого уровня этот диапазон может быть увеличен в несколько раз, что позволяет достигать очень высокой точности.

Достоверность – свойство информации быть правильно воспринятой.

Достоверность характеризуется вероятностью получения безошибочных результатов. Заданный уровень достоверности обеспечивается аппаратурно-программными средствами контроля самой ЭВМ. Возможны методы контроля достоверности путем решения эталонных задач и повторных расчетов. В особо ответственных случаях проводятся контрольные решения на других ЭВМ и сравнение результатов.

1.2. Классификация средств ЭВМ

В настоящее время в мире произведены, работают и продолжают выпускаться миллионы вычислительных машин, относящихся к различным поколениям, типам, классам, отличающихся своими областями применения, техническими характеристиками и вычислительными возможностями.

Классификация ЭВМ по принципу действия

Традиционно электронную вычислительную технику (ЭВТ) по принципу действия подразделяют на ***аналоговую, цифровую и гибридную***.

1. В ***аналоговых вычислительных машинах*** (АВМ) обрабатываемая информация представляется соответствующими значениями аналоговых величин: тока, напряжения, угла поворота какого-то механизма и т.п. Эти машины обеспечивают приемлемое быстродействие, но не очень высокую точность вычислений (0.001-0.01). Распространены подобные машины не очень широко. Они используются в основном в проектных и научно-исследовательских учреждениях в составе различных стендов по отработке сложных образцов техники. На АВМ наиболее эффективно решать математические задачи, содержащие дифференциальные уравнения, не требующие сложной логики. По своему назначению их можно рассматривать как *специализированные вычислительные машины*.
2. В настоящее время под словом ЭВМ обычно понимают ***цифровые вычислительные машины***, в которых информация кодируется двоичными

кодами чисел. Именно эти машины благодаря универсальным возможностям и являются самой массовой вычислительной техникой.

3. ***Гибридные вычислительные машины*** (ГВМ) – вычислительные машины комбинированного действия, работают с информацией, представленной и в цифровой, и в аналоговой форме; они совмещают в себе достоинства АВМ и ЦВМ. ГВМ целесообразно использовать для решения задач управления сложными быстродействующими техническими комплексами.

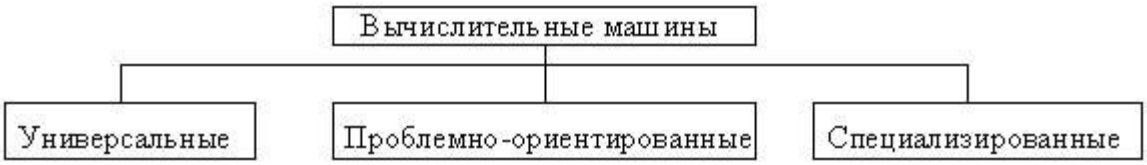
Рынок современных компьютеров отличается разнообразием и динамизмом, каких еще не знала ни одна область человеческой деятельности. Каждый год стоимость вычислений сокращается примерно на 25-30%, стоимость хранения единицы информации - до 40%. Практически каждое десятилетие меняется поколение машин, каждые два года - основные типы микропроцессоров - СБИС, определяющих характеристики новых ЭВМ. Такие темпы сохраняются уже многие годы.

То, что 10-15 лет назад считалось современной большой ЭВМ, в настоящее время является устаревшей техникой с очень скромными возможностями. Современный персональный компьютер с быстродействием в десятки и сотни миллионов операций в секунду становится доступным средством для массового пользователя.

В этих условиях любая предложенная классификация ЭВМ очень быстро устаревает и нуждается в корректировке. Например, в классификациях десятилетней давности широко использовались названия мини-, миди- и микроЭВМ, которые почти исчезли из обихода. Вместе с тем существует целый ряд закономерностей развития вычислительной техники, которые позволяют предвидеть и предсказывать основные результаты этого поступательного движения. Необходимо анализировать традиционные и новые области применения ЭВМ, классы и типы используемых вычислительных средств, сложившуюся конъюнктуру рынка информационных технологий и его динамику, количество и качество вычислительной техники, выпускаемой признанными лидерами - производителями средств ЭВТ и т.д. Коротко рассмотрим эти основные вопросы, выяснение которых позволит понять, какая вычислительная техника требуется для решения определенных задач.

Классификация ЭВМ по назначению

По назначению ЭВМ можно разделить на три группы**:** универсальные (общегоназначения), проблемно-ориентированные и специализированные.



Универсальные ЭВМ предназначены для решения самых различных инженерно-технических задач: экономических, математических, информационных и других задач, отличающихся сложностью алгоритмов и большим объемом обрабатываемых данных. Они широко используются в вычислительных центрах коллективного пользования и в других мощных вычислительных комплексах.

Характерными чертами *универсальных ЭВМ* являются:

- высокая производительность;
- разнообразие форм обрабатываемых данных: двоичных, десятичных, символьных, при большом диапазоне их изменения и высокой точности их представления;
- обширная номенклатура выполняемых операций, как арифметических, логических, так и специальных;
- большая емкость оперативной памяти;
- развитая организация системы ввода-вывода информации, обеспечивающая подключение разнообразных видов внешних устройств.

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; выполнением расчетов по относительно несложным алгоритмам; они обладают ограниченными по сравнению с универсальными ЭВМ аппаратными и программными ресурсами. К проблемно-ориентированным *ЭВМ* можно отнести, в частности, всевозможные управляющие вычислительные комплексы.

Специализированные ЭВМ используются для решения узкого круга задач или реализации строго определенной группы функций. Такая узкая ориентация ЭВМ позволяет четко специализировать их структуру, существенно снизить их сложность и стоимость при сохранении высокой производительности и надежности их работы.

К специализированным ЭВМ можно отнести, например, программируемые микропроцессоры специального назначения; адаптеры и контроллеры, выполняющие логические функции управления отдельными несложными техническими устройствами,

агрегатами и процессами; устройства согласования и сопряжения работы узлов вычислительных систем.

Академик В. М. Глушков указывал, что существуют три глобальные сферы деятельности человека, которые требуют использования качественно различных типов ЭВМ.

Первое направление является традиционным – применение ЭВМ для ***автоматизации вычислений.*** Научно-техническая революция во всех областях науки и техники постоянно выдвигает новые научные, инженерные, экономические задачи, которые требуют проведения крупномасштабных вычислений (задачи проектирования новых образцов техники, моделирования сложных процессов, атомная и космическая техника и др.). Отличительной особенностью этого направления является наличие хорошей математической основы, заложенной развитием математических наук и их приложений. Первые, а затем и последующие вычислительные машины классической структуры в первую очередь и создавались для автоматизации вычислений.

Вторая сфера применения ЭВМ связана с использованием их в ***системах управления сложными техническими системами***. Она родилась примерно в 60-е годы, когда ЭВМ стали интенсивно внедряться в контуры управления автоматических и автоматизированных систем. Математическая база этой новой сферы практически отсутствовала, в течение последующих 15-20 лет она была создана.

Новое применение вычислительных машин потребовало видоизменения их структуры. ЭВМ, используемые в управлении, должны были не только обеспечивать вычисления, но и *автоматизировать сбор данных и распределение результатов обработки*.

Сопряжение с каналами связи потребовало усложнения режимов работы ЭВМ, сделало их *многопрограммными и многопользовательскими*. Для исключения взаимных помех между программами пользователей в структуру машин были введены средства разграничения: блоки прерываний и приоритетов, блоки защиты и т.п. Для управления разнообразной периферией стали использоваться специальные процессоры ввода-вывода данных или каналы. Именно тогда и появился дисплей как средство оперативного человеко-машинного взаимодействия пользователя с ЭВМ.

Новой сфере работ в наибольшей степени отвечали мини-ЭВМ. Именно они стали использоваться для управления отраслями, предприятиями, корпорациями. Машины нового типа удовлетворяли следующим требованиям:

- ✓ были более дешевыми по сравнению с большими ЭВМ, обеспечивающими централизованную обработку данных;

- ✓ были более надежными, особенно при работе в контуре управления;
- ✓ обладали большой гибкостью и адаптируемостью настройки на конкретные условия функционирования;
- ✓ имели архитектурную прозрачность, т.е. структура и функции ЭВМ были понятны пользователям.

Начало выпуска подобных ЭВМ связано с малыми управляющими машинами PDP фирмы DEC. Термин «мини-ЭВМ» появился в 1968 г. применительно к модели PDP-8. В настоящее время использование мини-ЭВМ сокращается. Исчезает и термин мини-ЭВМ. На смену им приходят ЭВМ других типов: серверы, обеспечивающие диспетчерские функции в сетях ЭВМ, средние ЭВМ или старшие модели персональных ЭВМ (ПЭВМ).

Одновременно со структурными изменениями ЭВМ происходило и качественное изменение характера вычислений. Доля чисто математических расчетов постоянно сокращалась, и в настоящее время она составляет около 10% от всех вычислительных работ. Машины все больше стали использоваться для новых видов обработки: текстов, графики, звука и др.

Третье направление связано с применением ЭВМ для решения ***задач искусственного интеллекта***. Задачи искусственного интеллекта предполагают получение не точного результата, а чаще всего осредненного в статистическом, вероятностном смысле. Примеров подобных задач много:

- задачи робототехники,
- доказательства теорем,
- машинного перевода текстов с одного языка на другой,
- планирования с учетом неполной информации,
- составления прогнозов,
- моделирования сложных процессов и явлений и т.д.

Это направление все больше набирает силу. Во многих областях науки и техники создаются и совершенствуются базы данных (БД) и базы знаний (БЗ), экспертные системы (ЭС). Для технического обеспечения этого направления нужны качественно новые структуры ЭВМ с большим количеством вычислителей (ЭВМ или процессорных элементов), обеспечивающих параллелизм в вычислениях. По существу, ЭВМ уступают место сложнейшим вычислительным системам.

Уже это небольшое перечисление областей применения ЭВМ показывает, что для решения различных задач нужна соответственно и различная вычислительная техника. Поэтому рынок компьютеров постоянно имеет широкую градацию классов и моделей

ЭВМ. Фирмы-производители средств ВТ очень внимательно отслеживают состояние рынка ЭВМ. Они не просто констатируют отдельные факты и тенденции, а стремятся активно воздействовать на них и опережать потребности потребителей. Так, например, фирма IBM, выпускающая примерно 80% мирового машинного «парка», в настоящее время выпускает в основном *пять классов компьютеров*, перекрывая ими широкий класс задач пользователей

Классификация ЭВМ по размерам и функциональным возможностям

Можно предложить следующую классификацию средств вычислительной техники, в основу которой положено их разделение по быстродействию (см. рисунок 1.2):

- СуперЭВМ для решения крупномасштабных вычислительных задач, для обслуживания крупнейших информационных банков данных.
- Большие ЭВМ для комплектования ведомственных, территориальных и региональных вычислительных центров.
- Средние ЭВМ широкого назначения для управления сложными технологическими производственными процессами. ЭВМ этого типа могут использоваться и для управления распределенной обработкой информации в качестве сетевых серверов.
- Персональные и профессиональные ЭВМ, позволяющие удовлетворять индивидуальные потребности пользователей. На базе этого класса ЭВМ строятся автоматизированные рабочие места (АРМ) для специалистов различного уровня.
- Встраиваемые микропроцессоры, осуществляющие автоматизацию управления отдельными устройствами и механизмами.

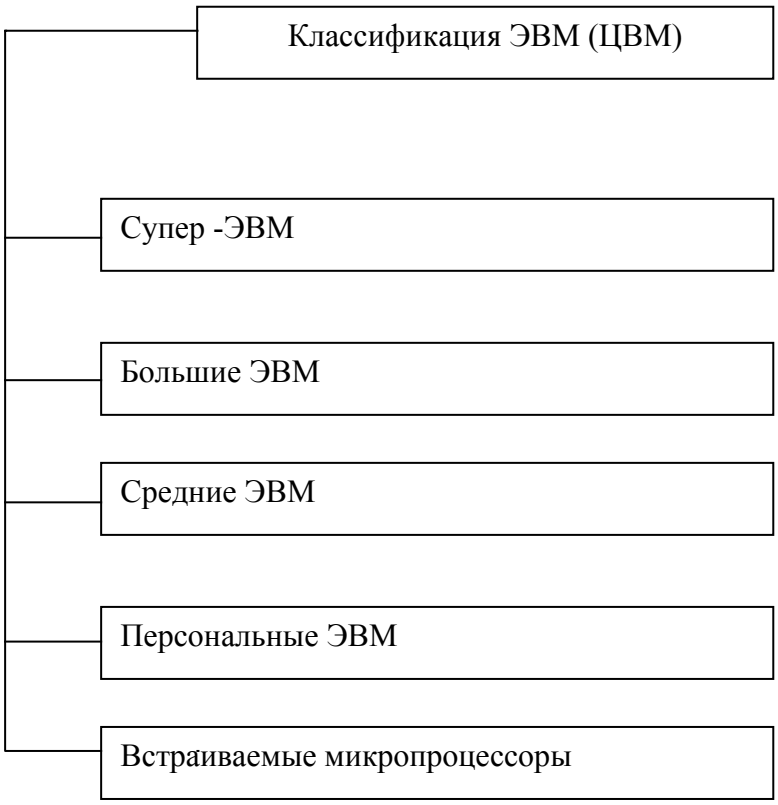


Рисунок 1.3 – Классификация ЭВМ

Функциональные возможности ЭВМ обусловливают важнейшие технико-эксплуатационные характеристики**:**

- быстродействие, измеряемое усредненным количеством операций, выполняемых машиной за единицу времени;
- разрядность и формы представления чисел, с которыми оперирует ЭВМ;
- номенклатура, емкость и быстродействие всех запоминающих устройств;
- номенклатура и технико-экономические характеристики внешних устройств хранения, обмена и ввода-вывода информации;
- типы и пропускная способность устройств связи и сопряжения узлов ЭВМ между собой (внутримашинного интерфейса);
- способность ЭВМ одновременно работать с несколькими пользователями и выполнять одновременно несколько программ (многопрограммность);
- типы и технико-эксплуатационные характеристики операционных систем, используемых в машине;
- наличие и функциональные возможности программного обеспечения;

- способность выполнять программы, написанные для других типов ЭВМ (программная совместимость с другими типами ЭВМ);
- система и структура машинных команд;
- возможность подключения к каналам связи и к вычислительной сети;
- эксплуатационная надежность ЭВМ;
- коэффициент полезного использования ЭВМ во времени, определяемый соотношением времени полезной работы и времени профилактики.

Большие ЭВМ (mainframe), которые представляют собой многопользовательские машины с центральной обработкой, с большими возможностями для работы с базами данных, с различными формами удаленного доступа. Казалось, что с появлением быстропрогрессирующих ПЭВМ большие ЭВМ обречены на вымирание. Однако они продолжают развиваться, и выпуск их снова стал увеличиваться, хотя их доля в общем парке постоянно снижается. По оценкам фирмы IBM, около половины всего объема данных в информационных системах мира должно храниться именно на больших машинах. Новое их поколение предназначено для использования в сетях в качестве крупных серверов. Начало этого направления было положено фирмой IBM еще в 60-е годы выпуском машин IBM/360, IBM/370. Эти машины получили широкое распространение в мире. Новая серия машин S/390 продолжает эту линию. Она насчитывает более двух десятков моделей: а) IBM S/390 Parallel Enterprise Server-Generation 3 (13 моделей) - призваны заменить большие ЭВМ ранних моделей. Они позволяют задавать переменную конфигурацию (число процессоров - 1-10, емкость оперативной памяти - 512-81292 Мбайта, число каналов - 3-256); б) IBM S/ 390 Multiprise 2000 (тоже 13 моделей) - ориентированы на использование на средних предприятиях (число процессоров 1-5).Развитие ЭВМ данного класса имеет большое значение для России. После подписания соглашения с фирмой IBM в марте 1993 г. Россия получила право производить 23 новейшие модели-аналоги ЭВМ IBM S/390 с производительностью от 1,5 до 167 млн. операций в секунду. По расходам на управление и эксплуатацию эти машины оказываются эффективнее других вычислительных средств. **Машины RS/6000** - очень мощные по производительности и предназначенные для построения рабочих станций для работы с графикой, Unix-серверов, кластерных комплексов. Первоначально эти машины предполагалось применять для обеспечения научных исследований.

Средние ЭВМ, предназначенные в первую очередь для работы в финансовых структурах (ЭВМ типа AS/400 (Advanced Portable Model 3) –«бизнес - компьютеры», 64-разрядные). В этих машинах особое внимание уделяется сохранению и безопасности

данных, программной совместимости и т.д. Они могут использоваться в качестве серверов в локальных сетях.

Компьютеры на платформе микросхем фирмы Intel. IBM-совместимые компьютеры этого класса составляют примерно 70% рынка всей компьютерной техники. Более половины их поступает в сферу малого бизнеса. Несмотря на столь внушительный объем выпуска персональных компьютеров этой платформы, фирма IBM проводит большие исследования и развитие собственной альтернативной платформы, получившей название Power PC. Это направление позволило бы значительно улучшить структуру аппаратурных средств ПК, а значит, и эффективность их применения. Однако новые модели этой платформы пока не выдерживают конкуренции с IBM PC. Немаловажным здесь является и неразвитость рынка программного обеспечения. Поэтому у массового пользователя это направление спроса не находит, и доля компьютеров с процессорами Power PC незначительна.

Кроме перечисленных типов вычислительной техники, необходимо отметить класс вычислительных систем, получивший название **«супер-ЭВМ»,** С развитием науки и техники постоянно выдвигаются новые крупномасштабные задачи, требующие выполнения больших объемов вычислений. Особенно эффективно применение суперЭВМ при решении задач проектирования, в которых натурные эксперименты оказываются дорогостоящими, недоступными или практически неосуществимыми. В этом случае ЭВМ позволяет методами численного моделирования получить результаты вычислительных экспериментов, обеспечивая приемлемое время и точность решения, т.е. решающим условием необходимости разработки и применения подобных ЭВМ является экономический показатель «производительность/стоимость». Например, при создании суперЭВМ GF-11 (Gigaflop-11) с быстродействием 11 млрд. операций в секунду предварительные расчеты, проведенные фирмой IBM, показали, что применение этой системы позволит решить целый комплекс новых задач. Одной из таких задач было уточнение массы протона на основе квантовой хромодинамики - доминирующей теории, пытающейся описать первичную структуру материи. При использовании новой ЭВМ должна была быть выполнена эта работа за 1,5 - 4 месяца с точностью 10%. Решение же этой задачи на существующей вычислительной технике требовало около 15 лет. Еще одним примером крупномасштабных задач следует считать задачу разработки новых схем СБИС для следующих поколений ЭВМ. СуперЭВМ позволяют по сравнению с другими типами машин точнее, быстрее и качественнее решать подобные задачи, обеспечивая необходимый приоритет в разработках перспективной вычислительной техники. Дальнейшее развитие суперЭВМ связывается с использованием направления массового

параллелизма, при котором одновременно могут работать сотни и даже тысячи процессоров. Образцы таких машин уже выпускаются несколькими фирмами:

nCube (гиперкубические ЭВМ),

Connection Machine,

Mass Par,

NCR/Teradata,

KSR,

ШМ RS/6000,

MPP и др.

Необходимо отметить и еще один класс наиболее массовых средств ЭВТ - **встраиваемые микропроцессоры**. Успехи микроэлектроники позволяют создавать миниатюрные вычислительные устройства, вплоть до однокристальных ЭВМ. Эти устройства, универсальные по характеру применения, могут встраиваться в отдельные машины, объекты, системы. Они находят все большее применение в бытовой технике (телефонах, телевизорах, электронных часах, микроволновых печах и т.д.), в городском хозяйстве (энерго-, тепло- , водоснабжении, регулировке движения транспорта и т.д.), на производстве (робототехнике, управлении технологическими процессами). Постепенно они входят в нашу жизнь, все больше изменяя среду обитания человека.

Высокие скорости вычислений, обеспечиваемые ЭВМ различных классов, позволяют перерабатывать и выдавать все большее количество информации, что, в свою очередь, порождает потребности в создании связей между отдельно используемыми ЭВМ. Поэтому все современные ЭВМ в настоящее время имеют средства подключения к сетям связи и комплексирования в системы.

Перечисленные типы ЭВМ, которые должны использоваться в индустриально развитых странах, образуют некое подобие пирамиды с определенным соотношением численности ЭВМ каждого слоя и набором их технических характеристик. Распределение вычислительных возможностей по слоям должно быть сбалансировано. Например, система обработки данных, используемая на Олимпийских играх в Атланте, содержала: 4 больших ЭВМ S/390,16 систем RS/6000, более 80 систем AS/400, более 7000 IBM PC, более 1000 лазерных принтеров, более 250 локальных сетей Token Ring и др. Многие ПЭВМ имели сопряжение с датчиками скорости, времени и т.д.

Требуемое количество суперЭВМ для отдельной развитой страны, такой, как Россия, должно составлять 100-200 шт., больших ЭВМ - тысячи, средних - десятки и сотни тысяч, ПЭВМ - миллионы, встраиваемых микроЭВМ - миллиарды. Все используемые ЭВМ различных классов образуют машинный парк страны, жизнедеятельность которого и его

информационное насыщение определяют успехи информатизации общества и научно-технического прогресса страны. Формирование сбалансированного машинного парка является сложной политической, экономической и социальной проблемой, решение которой требует многомиллиардных инвестиций. Для этого должна быть разработана соответствующая структура: создание специальных производств (элементной базы ЭВМ, программного обеспечения и технических связей), смена поколений машин и технологий, изменение форм экономического и административного управления, создание новых рабочих мест и т.д.

1.3. Общие принципы построения современных ЭВМ

Основным принципом построения всех современных ЭВМ является ***программное управление***. В основе его лежит представление алгоритма решения любой задачи в виде программы вычислений. Приведем определения основных понятий, связанных с программированием. Следует заметить, что строгого, однозначного определения алгоритма, равно как и однозначных методов его преобразования в программу вычислений, не существует.

> ***Алгоритм*** – заранее определенная последовательность действий, приводящая к решению задачи за конечное число шагов.

Алгоритм – конечный набор предписаний, определяющий решение задачи посредством конечного количества операций.[♦]

> ***Программа –*** это набор операторов (команд), который может быть представлен как единое целое в некоторой вычислительной системе и который используется для управления поведением этой системы

Программа (для ЭВМ) – упорядоченная последовательность команд, подлежащая обработке.[♦]

Принцип программного управления может быть осуществлен различными способами. Стандартом для построения практически всех ЭВМ стал **принцип**, описанный **Дж. фон Нейманом** в 1945 г. при построении еще первых образцов ЭВМ. Суть его заключается в следующем.

Все вычисления, предписанные алгоритмом решения задачи, должны быть представлены в виде программы, состоящей из последовательности управляющих слов-команд. Каждая команда содержит указания на конкретную выполняемую операцию, место нахождения (адреса) операндов и ряд служебных признаков.

[♦] стандарт ISO 2382/1-84

Операнды – переменные, значения которых участвуют в операциях преобразования данных. Список (массив) всех переменных (входных данных, промежуточных значений и результатов вычислений) является еще одним неотъемлемым элементом любой программы.

Для доступа к программам, командам и операндам используются их адреса. В качестве адресов выступают номера ячеек памяти ЭВМ, предназначенных для хранения объектов. Информация (командная и данные: числовая, текстовая, графическая и т.п.) кодируется двоичными цифрами 0 и 1. Поэтому различные типы информации, размещенные в памяти ЭВМ, практически неразличимы, идентификация их возможна лишь при выполнении программы, согласно ее логике, по контексту.

Каждый тип информации имеет форматы - структурные единицы информации, закодированные двоичными цифрами 0 и 1. Обычно все форматы данных, используемые в ЭВМ, кратны байту, т.е. состоят из целого числа байтов.

Последовательность битов в формате, имеющая определенный смысл, называется полем. Например, в каждой команде программы различают поле кода операций, поле адресов операндов. Применительно к числовой информации выделяют знаковые разряды, поле значащих разрядов чисел, старшие и младшие разряды.

Последовательность, состоящая из определенного принятого для данной ЭВМ числа байтов, называется ***словом***. Для больших ЭВМ размер слова составляет четыре байта, для ПЭВМ - два байта. В качестве структурных элементов информации различают также полуслово, двойное слово и др.

История развития ВТ (ЭВМ)

Рассмотрим развитие вычислительной техники с точки зрения структурной схемы построения и элементной базы.

1 поколение ЭВМ (1940-1955 г.г.)

ЭВМ этого поколения были построены на электронных лампах, запоминающие устройства (ЗУ) – на линиях задержки, для этого времени характерны ЗУ вращающегося типа и электростатические ЗУ (трубки Уильямса). В качестве устройств ввода-вывода (Увв) использовались перфоленты, перфокарты, магнитные ленты, печатающие устройства. Первые упоминания о результатах работ по созданию ЭВМ относятся к 1944-1946 г.г. С 1946 г. По июнь 1948 г. в Манчестерском университете (Великобритания) велись работы по созданию ЭВМ «Mark-1», которая в 1951 г. была выпущена в промышленное производство («Ferranti Mark 1»). С 1944г. по 1952 г. в Пенсильванском

университете (США) при участии фон Неймана была создана ЭВМ, в основу которой и был заложен принцип хранимой программы. С 1946 по 1949 г.г. в Кембриджском университете также велись работы по созданию ЭВМ.

2 поколение ЭВМ (1956-1960 г.г.)

ЭВМ этого поколения были построены на электронных лампах, транзисторах, запоминающие устройства построены на магнитных сердечниках, появились первые магнитные диски (магнитные барабаны), в это время начинают развиваться языки программирования, появились первые языки программирования высокого уровня (FORTRAN, AlgoL-60, Modula). Была создана фирма IBM.

Обобщенная структурная схема ЭВМ первых двух поколений представлена на рисунке 1.3.

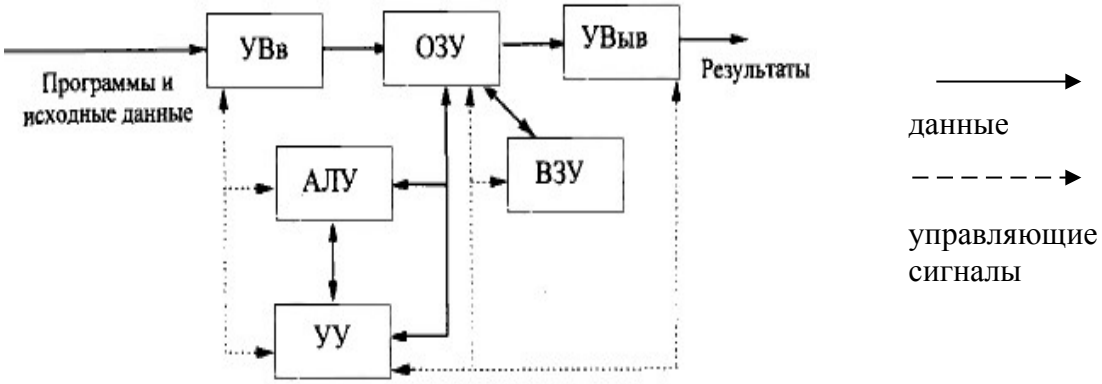


Рисунок 1.3 –Структурная схема ЭВМ первого и второго поколений

В любой ЭВМ имеются устройства ввода информации (**УВв**), с помощью которых пользователи вводят в ЭВМ программы решаемых задач и данные к ним. Введенная информация полностью или частично сначала запоминается в оперативном запоминающем устройстве (**ОЗУ**), а затем переносится во внешнее запоминающее устройство (**ВЗУ**), предназначенное для длительного хранения информации, где преобразуется в специальный программный объект - файл.

> ***Файл*** – хорошо структурированный набор данных (определенного формата (типа)), который хранится во внешней памяти компьютера и имеет собственное уникальное имя.

«***Файл*** - идентифицированная совокупность экземпляров полностью описанного в конкретной программе типа данных, находящихся вне программы во внешней памяти и доступных программе посредством специальных операций (ГОСТ 20866 - 85)».

При использовании файла в вычислительном процессе его содержимое переносится в ОЗУ. Затем программная информация команда за командой считывается в устройство управления (УУ). *Устройство управления* предназначается для автоматического выполнения программ путем принудительной координации всех остальных устройств ЭВМ. Цепи сигналов управления показаны на рисунке 1.3 штриховыми линиями. Вызываемые из ОЗУ команды дешифрируются устройством управления:

- определяется *код операции*, которую необходимо выполнить следующей,
- определяются *адреса операндов*, принимающих участие в данной операции.

В зависимости от количества используемых в команде операндов различаются одно-, двух-, трехадресные и безадресные команды. В одноадресных командах указывается, где находится один из двух обрабатываемых операндов. Второй операнд должен быть помещен заранее в арифметическое устройство (для этого в систему команд вводятся специальные команды пересылки данных между устройствами).

Двухадресные команды содержат указания о двух операндах, размещаемых в памяти (или в регистрах и памяти). После выполнения команды в один из этих адресов засылается результат, а находившийся там операнд теряется.

В трехадресных командах обычно два адреса указывают, где находятся исходные операнды, а третий - куда необходимо поместить результат.

В безадресных командах обычно обрабатывается один операнд, который до и после операции находится на одном из регистров арифметико-логического устройства (АЛУ). Кроме того, безадресные команды используются для выполнения служебных операций (очистить экран, заблокировать клавиатуру, снять блокировку и др.).

Все команды программы выполняются последовательно, команда за командой, в том порядке, как они записаны в памяти ЭВМ (естественный порядок следования команд). Этот порядок характерен для линейных программ, т.е. программ, не содержащих разветвлений. Для организации ветвлений используются команды, нарушающие естественный порядок следования команд. Отдельные признаки результатов r($r = 0$, $r < 0$, $r > 0$ и др.) устройство управления использует для изменения порядка выполнения команд программы.

Арифметико-логическое устройство (АЛУ) выполняет арифметические и логические операции над данными. Основной частью АЛУ является операционный автомат, в состав которого входят сумматоры, счетчики, регистры, логические преобразователи и др. Оно каждый раз перенастраивается на выполнение очередной операции. Результаты выполнения отдельных операций сохраняются для последующего

использования на одном из регистров АЛУ или записываются в память. Результаты, полученные после выполнения всей программы вычислений, передаются на устройства вывода (**УВыв**) информации. В качестве УВыв могут использоваться экран дисплея, принтер, графопостроитель и др.

Современные ЭВМ имеют достаточно развитые системы машинных операций. Например, ЭВМ типа IBM PC имеют около 200 различных операций (170 - 230 в зависимости от типа микропроцессора). Любая операция в ЭВМ выполняется по определенной микропрограмме, реализуемой в схемах АЛУ соответствующей последовательностью сигналов управления (микрокоманд). Каждая отдельная микрокоманда- это простейшее элементарное преобразование данных типа алгебраического сложения, сдвига, перезаписи информации и т.п.

Уже в первых ЭВМ для увеличения их производительности широко применялось совмещение операций. При этом последовательные фазы выполнения отдельных команд программы (формирование адресов операндов, выборка операндов, выполнение операции, отсылка результата) выполнялись отдельными функциональными блоками. В своей работе они образовывали своеобразный конвейер, а их параллельная работа позволяла обрабатывать различные фазы целого блока команд. Этот принцип получил дальнейшее развитие в ЭВМ следующих поколений. Но все же первые ЭВМ имели очень сильную централизацию управления, единые стандарты форматов команд и данных, «жесткое» построение циклов выполнения отдельных операций, что во многом объясняется ограниченными возможностями используемой в них элементной базы. Центральное ***устройство управления (УУ)*** обслуживало не только вычислительные операции, но и операции ввода-вывода, пересылок данных между ЗУ и др. Все это позволяло в какой-то степени упростить аппаратуру ЭВМ, но сильно сдерживало рост их производительности.

3 поколение ЭВМ (1960-1970 г.г.)

В связи с появлением интегральных микросхем, транзисторов, диодов и других устройств, появилась возможность организовывать прерывания в вычислениях. Это позволило организовать мультипрограммный режим работы, а появление операционных систем (ОС) дало качественно новые возможности для управления вычислительным процессом. Первой торговой маркой фирмы IBM стала ОС DOS, а затем появилась ОС MS DOS.

> ***Операционная система (ОС)*** – комплекс взаимосвязанных файлов и программ, которые отвечают за управление вычислительным процессом на компьютере и за организацию взаимодействие пользователя и компьютера.

В ЭВМ третьего поколения произошло усложнение структуры за счет разделения процессов ввода-вывода информации и ее обработки (рисунок 1.4).

Сильносвязанные устройства АЛУ и УУ получили название ***процессор***, т.е. устройство, предназначенное для обработки данных. В схеме ЭВМ появились также дополнительные устройства, которые имели названия: процессоры ввода-вывода, устройства управления обменом информацией, каналы ввода-вывода (**КВВ**). Последнее название получило наибольшее распространение применительно к большим ЭВМ. Здесь наметилась тенденция к децентрализации управления и параллельной работе отдельных устройств, что позволило резко повысить быстродействие ЭВМ в целом.

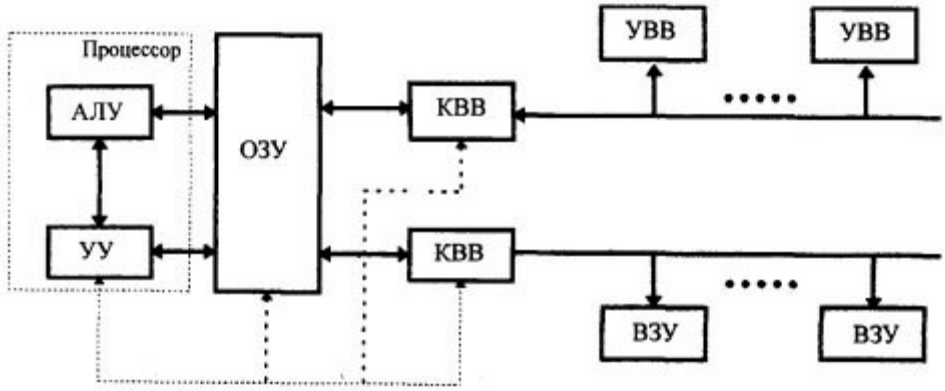


Рисунок 1.4 – Структурная схема ЭВМ третьего поколения

Среди каналов ввода-вывода выделяли ***мультиплексные каналы***, способные обслуживать большое количество медленно работающих устройств ввода-вывода (УВВ), и ***селекторные каналы***, обслуживающие в многоканальных режимах скоростные внешние запоминающие устройства (ВЗУ).

4 поколение ЭВМ (1970-1990 г.г.)

В начале 70-х годов за счет развития элементной базы (появление больших интегральных схем (БИС), магнитных устройств большой емкости, микропроцессоров) стоимость хранения единицы информации на магнитных носителях стала меньше, чем на традиционных носителях (бумажных).Это привело к созданию персональных ЭВМ (ПЭВМ) и позволило внедрять их во все сферы производства. Кроме того, появление новых мощных языков программирования позволило создавать большие информационные системы, ориентированные на обработку больших объемов данных (БД).

> ***База данных (БД)*** – один или несколько взаимосвязанных фалов, в которых хранится информация о реальных объектах конкретной предметной области в хорошо структурированном виде.

В персональных ЭВМ, относящихся к ЭВМ четвертого поколения, произошло дальнейшее изменение структуры (рисунок 1.5). Они унаследовали ее от мини-ЭВМ.

Соединение всех устройств в единую машину обеспечивается с помощью общей системной шины, представляющей собой линии передачи данных, адресов, сигналов управления и питания. Единая система аппаратурных соединений значительно упростила структуру, сделав ее еще более децентрализованной. Все передачи данных по шине осуществляются под управлением сервисных программ.

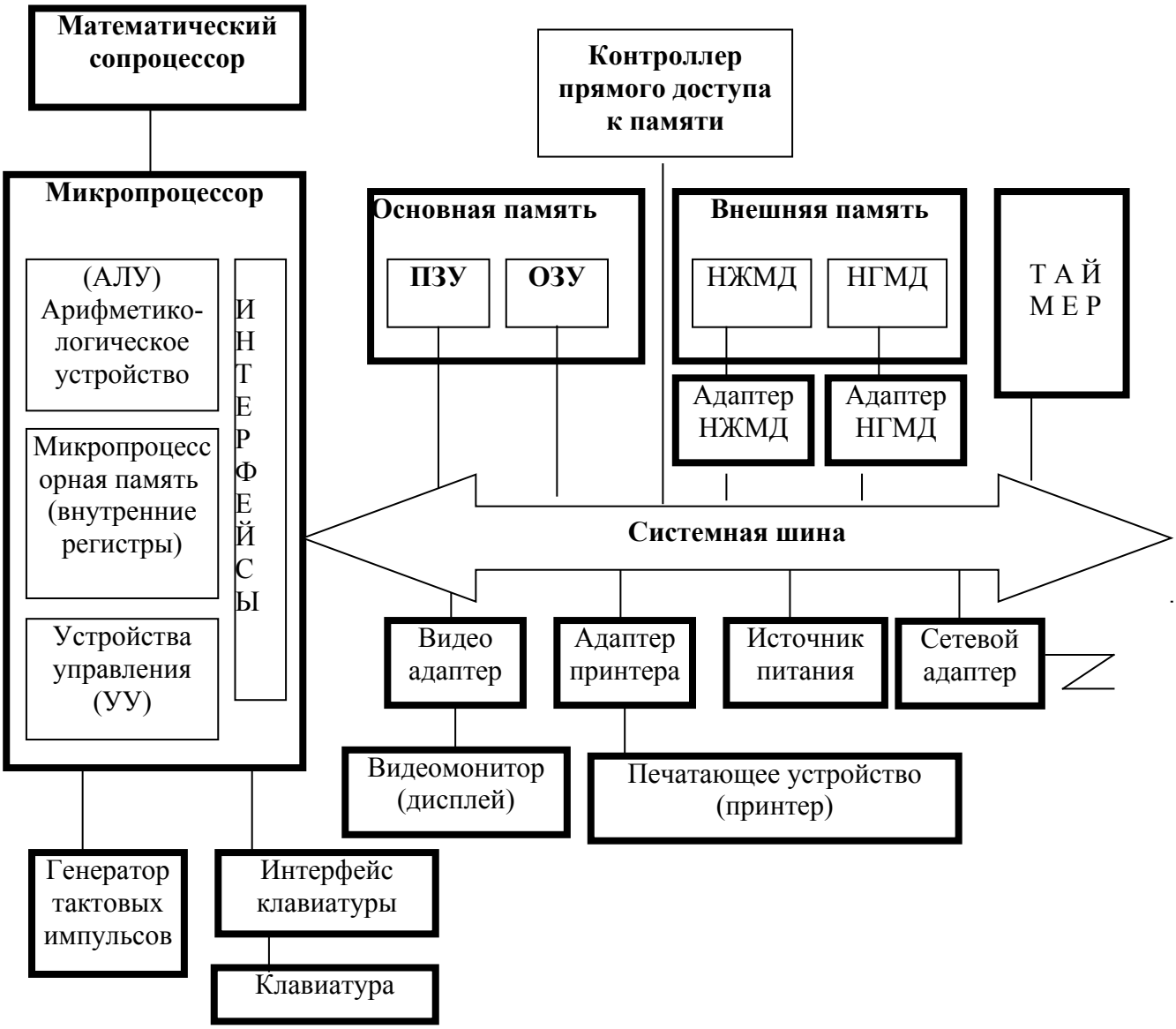


Рисунок 1.5 - Структурная схема ПЭВМ 4-го поколения

Ядро ПЭВМ образуют *процессор* и *основная память* (**ОП**), состоящая из оперативной памяти и постоянного запоминающего устройства (ПЗУ). ПЗУ предназначается для записи и постоянного хранения наиболее часто используемых программ управления и справочной информации, позволяет оперативно только считывать хранящуюся в нем информацию (изменить информацию в ПЗУ нельзя).

Микропроцессор (МП) – центральный блок ПК, предназначенный для управления работой всех блоков машины и для преобразования информации, т.е. выполнения арифметических и логических операций над информацией.

В состав МП входят следующие устройства:

- *арифметико-логическое устройство* (АЛУ), которое предназначено для выполнения всех арифметических и логических операций над числовой и символьной информацией, для ускорения операций с вещественными числами. К АЛУ подключается математический сопроцессор. Математический сопроцессор используется для ускоренного выполнения операций над числами с плавающей запятой, имеет свою систему команд и работает параллельно с основным МП, но под его управлением. Последние модели МП, начиная с МП 80486DX, включают сопроцессор в свой состав в качестве устройства с плавающей точкой.;
- *внутренние регистры*, которые служат для кратковременного хранения, записи и выдачи информации, непосредственно используемой в вычислениях в ближайшее время работы ПК. Регистры – быстродействующие ячейки памяти различной длины (8÷ 64 разрядов);
- *устройство управления* (УУ), которое формирует и подает на все блоки машины управляющие символы, формирует адреса ячеек памяти и передает их для выполнения операции;
- *интерфейс МП*, который реализует сопряжение и связь с другими устройствами ПК; включает в себя связь между устройствами МП, буферные регистры, схемы управления портами ввода-вывода и системной шиной. Интерфейс – совокупность средств сопряжения и связи устройств компьютера, обеспечивающая их эффективное взаимодействие.

Системная шина (СШ) – основное устройство, обеспечивающее объединение в единое целое всех устройств ЭВМ их взаимодействие.

СШ включает в себя четыре шины:

- *шину данных*, содержащую провода и схемы сопряжения для параллельной передачи всех разрядов числового кода операнда;
- *шину адреса*, содержащую провода и схемы сопряжения для параллельной передачи всех разрядов числового кода адреса ячейки основной памяти или порта ввода-вывода внешнего устройства;
- *шину управления*, содержащую провода и схемы сопряжения для передачи управляющих сигналов во все блоки ПК;
- *шину питания*, имеющую провода и схемы сопряжения для подключения блоков ПК к системе электропитания.

Системная шина обеспечивает три направления передачи информации:

- между МП и ОП;
- между МК и портами ввода-вывода внешних устройств;
- между ОП и портами ввода-вывода внешних устройств (в режиме прямого доступа к памяти).

Все блоки ПК (точнее их порты ввода-вывода) через соответствующие унифицированные разъемы подключаются к шине единообразно: непосредственно или через соответствующие ***адаптеры*** – согласователи скоростей работы сопрягаемых устройств или ***контроллеры*** - специальные устройства управления периферийной аппаратурой. Контроллеры в ПЭВМ играют роль каналов ввода-вывода. В качестве особых устройств следует выделить таймер - устройство измерения времени и контроллер прямого доступа к памяти (**КПД**) - устройство, обеспечивающее доступ к ОП, минуя процессор.

Управление системной шиной осуществляется либо непосредственно, либо через контроллер шины. Обмен информацией между внешними устройствами и системной шиной выполняется с использованием ASCII-кодов.

Устройства ввода информации:

- клавиатура;
- графические планшеты (диджитайзеры) – ручной ввод графической информации путем перемещения по планшету специального указателя (пера);
- сканеры;
- манипуляторы (джойстик – рычаг, мышь, трекбол – шар в оправе, световое перо – для ввода графической информации на экран дисплея путем управления движением курсора по экрану);
- сенсорные экраны – для ввода отдельных элементов изображения, программ или команд в ПК;
- устройство речевого ввода-вывода – быстроразвивающиеся средства мультимедиа;
- видеомонитор – устройство для отображения вводимой и выводимой информации.

Устройства вывода информации:

- принтеры – печатающие устройства для регистрации информации на бумажный носитель.
- графопостроители (плоттеры) – устройства для вывода графической информации из ПК на бумажный носитель. Векторные плоттеры – вычерчивают изображение при помощи пера. Растровые бывают электростатические, струйные и лазерные. По конструкции плоттеры подразделяются на планшетные и барабанные.

Устройства связи и телекоммуникации используются для связи с другими приборами и подключения ПК к каналам связи и другим ЭВМ и вычислительным сетям (сетевые интерфейсные платы, мультиплексоры передачи данных, модемы).

Сетевой адаптер служит для подключения ПК к каналу связи для работы в составе вычислительной сети. В глобальных сетях функции сетевого адаптера выполняет модулятор-демодулятор (модем).

Средства мультимедиа (multimedia – многосредовость) – комплекс аппаратных и программных средств, позволяющих человеку общаться с ПК, используя самые разные, естественные для себя среды: звук, видео, графику, тексты, анимацию и др. К средствам мультимедиа относятся: устройства речевого ввода-вывода, сканеры, высококачественные видео- и звуковые платы, платы видео-захвата (снимают изображения с видеомагнитофона или видеокамеры и вводят его в ПК), высококачественные акустические и видеовоспроизводящие системы с усилителями, звуковыми колонками, большими видео-экранами. К средствам мультимедиа относят также внешние запоминающие устройства большой емкости на оптических дисках, часто используемые для записи звуковой и видео информации.

Генератор тактовых импульсов **(ГТИ)** генерирует последовательность электрических импульсов; синхронизирующих работу всех устройств ПК; частота генерируемых импульсов определяет тактовую частоту ПК, которая является одной из основных *характеристик ПЭВМ.*

Таймер **–** внутримашинные электронные часы, обеспечивающие автоматический съем значения текущего момента времени (год, месяц, день, часы, минуты, секунды и доли секунды). Таймер подключается к автономному источнику питания – аккумулятору и при отключении ПК от сети продолжает работать.

Способ формирования структуры ПЭВМ является достаточно логичным и естественным стандартом для данного класса ЭВМ. Децентрализация построения и управления вызвала к жизни такие элементы, которые являются общим стандартом структур современных ЭВМ:

5 поколение ЭВМ (1990-2000 г.г.)

6 поколение ЭВМ (2000 г. по н/вр)

Появилась возможность распараллеливания вычислительного процесса за счет использования 2-х или более процессоров, объединенных в вычислительное ядро. Появилась всемирная сеть Интернет, обеспечивающая потенциальный доступ любого компьютера к глобальной информации. Стали бурно развиваться телекоммуникационные

технологии: появилась возможность передавать большие объемы информации на большие расстояния (радио-, спутниковая связь). Так информация хранится на разных ЭВМ, то появилась необходимость разработки концепции распределенной обработки и хранения данных и алгоритмов параллельных вычислений, позволяющих увеличить быстродействие систем обработки информации. Нейро-сети и нейро-технологии позволили это сделать.

Модульность построения, магистральность, иерархия управления

Модульность построения предполагает выделение в структуре ЭВМ достаточно автономных, функционально и конструктивно законченных устройств (процессор, модуль памяти, накопитель на жестком или гибком магнитном диске).

Модульная конструкция ЭВМ делает ее открытой системой, способной к адаптации и совершенствованию. К ЭВМ можно подключать дополнительные устройства, улучшая ее технические и экономические показатели. Появляется возможность увеличения вычислительной мощности, улучшения структуры путем замены отдельных устройств на более совершенные, изменения и управления конфигурацией системы, приспособления ее к конкретным условиям применения в соответствии с требованиями пользователей.

В современных ЭВМ принцип децентрализации и параллельной работы распространен как на периферийные устройства, так и на сами ЭВМ (процессоры). Появились вычислительные системы, содержащие несколько вычислителей (ЭВМ или процессоры), работающие согласованно и параллельно. Внутри самой ЭВМ произошло еще более резкое разделение функций между средствами обработки. Появились отдельные специализированные процессоры, например сопроцессоры, выполняющие обработку чисел с плавающей точкой, матричные процессоры и др.

Все существующие типы ЭВМ выпускаются семействами, в которых различают старшие и младшие модели. Всегда имеется возможность замены более слабой модели на более мощную. Это обеспечивается информационной, аппаратурной и программной совместимостью. Программная совместимость в семействах устанавливается по принципу снизу-вверх, т.е. программы, разработанные для ранних и младших моделей, могут обрабатываться и на старших, но не обязательно наоборот.

Модульность структуры ЭВМ требует стандартизации и унификации оборудования, номенклатуры технических и программных средств, средств сопряжения - интерфейсов, конструктивных решений, унификации типовых элементов замены, элементной базы и нормативно-технической документации. Все это способствует

улучшению технических и эксплуатационных характеристик ЭВМ, росту технологичности их производства.

Децентрализация управления предполагает ***иерархическую организацию структуры ЭВМ***. Централизованное управление осуществляет устройство управления главного, или центрального, процессора. Подключаемые к центральному процессору модули (контроллеры и КВВ) могут, в свою очередь, использовать специальные шины или магистрали для обмена управляющими сигналами, адресами и данными. Инициализация работы модулей обеспечивается по командам центральных устройств, после чего они продолжают работу по собственным программам управления. Результаты выполнения требуемых операций представляются ими “вверх по иерархии” для правильной координации всех работ.

Иерархический принцип построения и управления характерен не только для структуры ЭВМ в целом, но и для отдельных ее подсистем. Например, по этому же принципу строится система памяти ЭВМ.

Так, с точки зрения пользователя желательно иметь в ЭВМ оперативную память большой информационной емкости и высокого быстродействия. Однако одноуровневое построение памяти не позволяет одновременно удовлетворять этим двум противоречивым требованиям. Поэтому память современных ЭВМ строится по многоуровневому, пирамидальному принципу.

Иерархический принцип построения памяти

> ***Памятью ЭВМ*** называется совокупность устройств, служащих для запоминания, хранения и выдачи информации. Отдельные устройства, входящие в эту совокупность, называются запоминающими устройствами (ЗУ) того или иного типа.

Термин «запоминающее устройство» обычно используется, когда речь идет о принципе построения некоторого устройства памяти (например, полупроводниковое ЗУ, ЗУ на жестком магнитном диске и т.п.), а термин «память» - когда хотят подчеркнуть выполняемую устройством памяти логическую функцию или место расположения в составе оборудования ЭВМ (например, оперативная память - ОП, внешняя память и т.п.). В тех вопросах, где эти отличия не имеют принципиального значения, термины «память» и «запоминающее устройство» мы будем использовать как синонимы.

Запоминающие устройства играют важную роль в общей структуре ЭВМ. По некоторым оценкам производительность компьютера на разных классах задач на 40-50% определяется характеристиками ЗУ различных типов, входящих в его состав.

К основным параметрам, характеризующим запоминающие устройства, относятся ***емкость*** и ***быстродействие***.

> ***Емкость памяти*** - это максимальное количество данных, которое в ней может храниться.

Емкость запоминающего устройства измеряется количеством адресуемых элементов (ячеек) ЗУ и длиной ячейки в битах. В настоящее время практически все запоминающие устройства в качестве минимально адресуемого элемента используют 1 байт. Поэтому емкость памяти обычно определяется в байтах, килобайтах, мегабайтах, гигабайтах и т.д.

За одно обращение к запоминающему устройству производится считывание или запись некоторой единицы данных, называемой *словом*, различной для устройств разного типа. Это определяет разную организацию памяти. Например, память объемом 1 мегабайт может быть организована как 1М слов по 1 байту, или 512К слов по 2 байта каждое, или 256К слов по 4 байта и т.д.

В то же время, в каждой ЭВМ используется свое понятие машинного слова, которое применяется при определении архитектуры компьютера, в частности при его программировании, и не зависит от размерности слова памяти, используемой для построения данной ЭВМ. Например, компьютеры с архитектурой IBM PC имеют машинное слово длиной 2 байта.

Быстродействие памяти определяется продолжительностью операции обращения, то есть временем, затрачиваемым на поиск нужной информации в памяти и на ее считывание, или временем на поиск места в памяти, предназначаемого для хранения данной информации, и на ее запись:

$$t_{обр} = \max (t_{обр\ сч}, t_{обр\ зп})$$

где $t_{обр\ сч}$ - быстродействие ЗУ при считывании информации; $t_{обр\ зп}$ - быстродействие ЗУ при записи.

Запоминающие устройства можно классифицировать по целому ряду параметров и признаков. На рисунке 1.6 представлена классификация по типу обращения и организации доступа к ячейкам ЗУ.

Рисунок 1.6 - Классификация запоминающих устройств

По типу обращения *ЗУ* делятся на устройства, допускающие как чтение, так и запись информации, и постоянные *запоминающие устройства* (ПЗУ), предназначенные только для чтения записанных в них данных (***ROM*** - **read only memory**). *ЗУ* первого типа используются в процессе работы процессора для хранения выполняемых программ, исходных данных, промежуточных и окончательных результатов. В ПЗУ, как правило, хранятся системные программы, необходимые для запуска компьютера в работу, а также константы. В некоторых ЭВМ, предназначенных, например, для работы в системах управления по одним и тем же неизменяемым алгоритмам, все программное обеспечение может храниться в ПЗУ.

В ***ЗУ с произвольным доступом*** (***RAM*** - **random access memory**) время доступа не зависит от места расположения участка памяти (например, *ОЗУ*).

В ***ЗУ с прямым (циклическим) доступом*** благодаря непрерывному вращению носителя информации (например, магнитный диск - МД) возможность обращения к некоторому участку носителя циклически повторяется. Время доступа здесь зависит от взаимного расположения этого участка и головок чтения/записи и во многом определяется скоростью вращения носителя.

В ***ЗУ с последовательным доступом*** производится последовательный просмотр участков носителя информации, пока нужный участок не займет некоторое нужное положение напротив головок чтения/записи (например, магнитные ленты - МЛ).

Как отмечалось выше, основные характеристики *запоминающих устройств* - это *емкость* и *быстродействие*. Идеальное *запоминающее устройство* должно обладать бесконечно большой *емкостью* и иметь бесконечно малое время обращения. На практике эти параметры находятся в противоречии друг другу: в рамках одного типа *ЗУ* улучшение одного из них ведет к ухудшению значения другого. К тому же следует иметь в виду и

экономическую целесообразность построения *запоминающего устройства* с теми или иными характеристиками при данном уровне развития технологии. Поэтому в настоящее время *запоминающие устройства* компьютера, как это и предполагал Нейман, строятся по *иерархическому принципу* (рисунок 1.7).

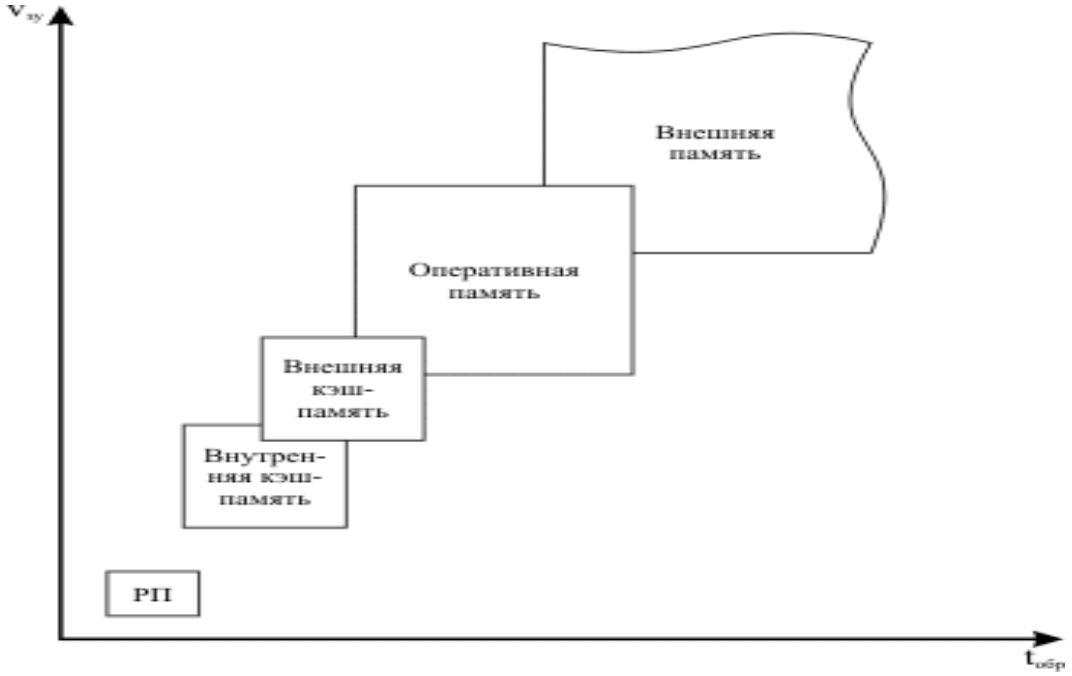


Рисунок 1.7 - Иерархическая организация памяти в современных ЭВМ

Иерархическая структура памяти позволяет экономически эффективно сочетать хранение больших объемов информации с быстрым доступом к информации в процессе ее обработки.

На нижнем уровне иерархии находится ***регистровая память*** - набор регистров, входящих непосредственно в состав микропроцессора (центрального процессора - CPU). Регистры CPU программно доступны и хранят информацию, наиболее часто используемую при выполнении программы: промежуточные результаты, составные части адресов, счетчики циклов и т.д. *Регистровая память* имеет относительно небольшой объем (до нескольких десятков машинных слов). РП работает на частоте процессора, поэтому время доступа к ней минимально. Например, при частоте работы процессора 2 ГГц время обращения к его регистрам составит всего 0,5 нс.

Оперативная память - устройство, которое служит для хранения информации (программ, исходных данных, промежуточных и конечных результатов обработки), непосредственно используемой в ходе выполнения программы в процессоре. В настоящее время объем ОП персональных компьютеров составляет несколько сотен мегабайт. *Оперативная память* работает на частоте системной шины и требует 6-8 циклов синхронизации шины для обращения к ней. Так, при частоте работы системной шины 100

МГц (при этом период равен 10 нс) время обращения к *оперативной памяти* составит несколько десятков наносекунд.

Для заполнения пробела между РП и ОП по объему и времени обращения в настоящее время используется ***кэш-память***, которая организована как более быстродействующая (и, следовательно, более дорогая) статическая *оперативная память* со специальным механизмом записи и считывания информации и предназначена для хранения информации, наиболее часто используемой при работе программы (предназначена для хранения активных страниц объемом десятки и сотни Кбайтов). Как правило, часть *кэш-памяти* располагается непосредственно на кристалле микропроцессора (внутренний *кэш*), а часть - вне его (внешняя *кэш-память*). *Кэш-память* программно недоступна. Для обращения к ней используются аппаратные средства процессора и компьютера.

Внешняя память организуется, как правило, на магнитных и оптических дисках, магнитных лентах. Емкость дисковой памяти достигает десятков гигабайт при времени обращения менее 1 мкс. Магнитные ленты вследствие своего *малого быстродействия* и *большой емкости* используются в настоящее время в основном только как устройства резервного копирования данных, обращение к которым происходит редко, а может быть и никогда. Время обращения для них может достигать нескольких десятков секунд.

Следует отметить, что электронная вычислительная техника развивается чрезвычайно быстрыми темпами. Так, согласно эмпирическому «закону Мура», производительность компьютера удваивается приблизительно каждые 18 месяцев. Поэтому все приводимые в данном пособии количественные характеристики служат по большей части только для отражения основных соотношений и тенденций в развитии тех или иных компонентов и устройств компьютеров.

Часть машинных программ, обеспечивающих автоматическое управление вычислениями и используемых наиболее часто, может размещаться в постоянном запоминающем устройстве (ПЗУ). На более низких уровнях иерархии находятся внешние запоминающие устройства на магнитных носителях:

- на жестких и гибких магнитных дисках,
- магнитных лентах,
- магнитооптических дисках и др.

Их отличает более низкое быстродействие и очень большая емкость.

Организация заблаговременного обмена информационными потоками между ЗУ различных уровней при децентрализованном управлении ими позволяет рассматривать

иерархию памяти как единую абстрактную кажущуюся (виртуальную) память. Согласованная работа всех уровней обеспечивается под управлением программ *операционной системы*. Пользователь имеет возможность работать с памятью, намного превышающей емкость ОЗУ.

Децентрализация управления и структуры ЭВМ позволила перейти к более сложным многопрограммным (мультипрограммным) режимам. При этом в ЭВМ одновременно может обрабатываться несколько программ пользователей.

В ЭВМ, имеющих один процессор, многопрограммная обработка является кажущейся. Она предполагает параллельную работу отдельных устройств, задействованных в вычислениях по различным задачам пользователей. Например, компьютер может производить распечатку каких-либо документов и принимать сообщения, поступающие по каналам связи. Процессор при этом может производить обработку данных по третьей программе, а пользователь - вводить данные или программу для новой задачи, слушать музыку и т.п.

В ЭВМ или вычислительных системах, имеющих несколько процессоров обработки, многопрограммная работа может быть более глубокой. Автоматическое управление вычислениями предполагает усложнение структуры за счет включения в ее состав систем и блоков, разделяющих различные вычислительные процессы друг от друга, исключающие возможность возникновения взаимных помех и ошибок (системы прерываний и приоритетов, защиты памяти). Самостоятельного значения в вычислениях они не имеют, но являются необходимым элементом структуры для обеспечения этих вычислений.

Как видно, полувековая история развития ЭВТ дала не очень широкий спектр основных структур ЭВМ. Все приведенные структуры не выходят за пределы классической структуры фон Неймана. Их объединяют следующие Традиционные признаки:

- **ядро ЭВМ образует процессор** - единственный вычислитель в структуре, дополненный каналами обмена информацией и памятью;
- **линейная организация ячеек всех видов памяти** фиксированного размера;
- **одноуровневая адресация ячеек памяти**, стирающая различия между всеми типами информации;
- **внутренний машинный язык низкого уровня**, при котором команды содержат элементарные операции преобразования простых операндов;
- **последовательное централизованное управление вычислениями**;

- достаточно примитивные возможности устройств ввода-вывода. Несмотря на все достигнутые успехи, классическая структура ЭВМ не обеспечивает возможностей дальнейшего увеличения производительности. Наметился кризис, обусловленный рядом существенных недостатков:
  - плохо развитые средства обработки нечисловых данных (структуры, символы, предложения, графические образы, звук, очень большие массивы данных и др.);
  - несоответствие машинных операций операторам языков высокого уровня;
  - примитивная организация памяти ЭВМ;
  - низкая эффективность ЭВМ при решении задач, допускающих параллельную обработку и т.п.

Все эти недостатки приводят к чрезмерному усложнению комплекса программных средств, используемого для подготовки и решения задач пользователей.

В ЭВМ будущих поколений, с использованием в них «встроенного искусственного интеллекта», предполагается дальнейшее усложнение структуры; В-первую очередь это касается совершенствования процессов общения пользователей с ЭВМ (использование аудио-, видеоинформации, систем мультимедиа и др.) , обеспечения доступа к базам данных и базам знаний, организации параллельных вычислений. Несомненно, что этому должны соответствовать новые параллельные структуры с новыми принципами их построения. В качестве примера укажем, что самая быстрая ЭВМ фирмы IBM в настоящее время обеспечивает быстродействие 600 MIPS (миллионов команд в секунду), самая же большая гиперкубическая система nCube дает быстродействие 123.103 MBPS. Расчеты показывают, что стоимость одной машинной операции в гиперсистеме примерно в тысячу раз меньше. Вероятно, подобными системами будут обслуживаться большие информационные хранилища.

1.4. Функции программного обеспечения

Электронные вычислительные машины являются универсальными техническими средствами автоматизации вычислительных работ, т.е. они способны решать любые задачи, связанные с преобразованием информации. Однако подготовка задач к решению на ЭВМ была и остается до настоящего времени достаточно трудоемким процессом, требующим от пользователей во многих случаях специальных знаний и навыков.

Для снижения трудоемкости подготовки задач к решению, более эффективного использования отдельных технических, программных средств и ЭВМ в целом, а также

облегчения их эксплуатации каждая ЭВМ имеет специальный комплекс программных средств регулярного применения. Эти средства обеспечивают взаимодействие пользователей с ЭВМ и являются своеобразным “посредником” между ними. Они получили название программного обеспечения (ПО) ЭВМ.

> Под ***программным обеспечением*** будем понимать *комплекс программных средств регулярного применения*, предназначенный для подготовки и решения задач пользователей.

Программное обеспечение отдельных ЭВМ и ВС может сильно различаться составом используемых программ, который определяется классом используемой вычислительной техники, режимами ее применения, содержанием вычислительных работ пользователей и т.п. Развитие ПО современных ЭВМ и ВС в значительной степени носит эволюционный и эмпирический характер, но можно выделить закономерности в его построении.

В общем случае процесс подготовки и решения задач на ЭВМ пользователями предусматривает выполнение следующей последовательности этапов (рисунок 1.8):

Рисунок 1.8 - Автоматизация подготовки и решения задач в ЭВМ

- формулировка проблемы и математическая постановка задачи;
- выбор метода и разработка алгоритма решения;
- программирование (запись алгоритма) с использованием некоторого алгоритмического языка;

- планирование и организация вычислительного процесса - порядка и последовательности использования ресурсов ЭВМ и ВС;
- формирование «машинной программы», т.е. программы, которую непосредственно будет выполнять ЭВМ;
- собственно решение задачи - выполнение вычислений по готовой программе.

По мере развития вычислительной техники автоматизация этих этапов идет снизу-вверх. В ЭВМ 1-го поколения автоматизации подлежал только последний этап. Все пять предыдущих этапов пользователь должен был готовить вручную самостоятельно. Трудоемкий и рутинный характер этих работ был источником большого количества ошибок в заданиях. Поэтому в ЭВМ следующих поколений появились сначала элементы, а затем целые системы, облегчающие процесс подготовки задач к решению.

Для ЭВМ 2-го поколения характерно широкое применение алгоритмических языков (Автокоды, Алгол, Фортран и др.) и соответствующих трансляторов, позволяющих автоматически формировать машинные программы по их описанию на алгоритмическом языке. Здесь же широко стали внедряться библиотеки стандартных программ, что позволило строить машинные программы блоками, используя накопленный и приобретенный программистами опыт. Отметим, что временные границы появления всех нововведений достаточно размыты. Обычно их истоки можно обнаружить в недрах ЭВМ предыдущих поколений.

ЭВМ 3-го поколения характеризуются расцветом операционных систем (ОС), отвечающих за организацию и управление вычислительным процессом. Именно здесь слово “ЭВМ” все чаще стало заменяться понятием “вычислительная система”, что в большей степени отражало усложнение как аппаратурной, так и программной частей ЭВМ. Стоимость программного обеспечения стала расти и в настоящее время намного опережает стоимость аппаратных средств (рисунок 1.9).

Рисунок 1.9 – Динамика изменения стоимости аппаратурных и программных средств

Операционная система планирует последовательность распределения и использования ресурсов вычислительной системы, а также обеспечивает их согласованную работу. Под ресурсами обычно понимают те средства, которые используются для вычислений: машинное время отдельных процессоров или ЭВМ, входящих в систему; объемы оперативной и внешней памяти; отдельные устройства, информационные массивы; библиотеки программ; отдельные программы как общего, так и специального применения и т.п. Наиболее употребительные функции ОС в части обработки внештатных ситуаций (защита программ от взаимных помех, системы прерываний и приоритетов, служба времени, сопряжение с каналами связи и т.д.) были полностью или частично реализованы аппаратно. Одновременно были реализованы более сложные режимы работы: коллективный доступ к ресурсам, мультипрограммные режимы. Часть этих решений стала своеобразным стандартом и начала использоваться повсеместно в ЭВМ различных классов. Это позволило в значительной степени повысить эффективность применения ЭВМ и ВС в целом.

В ЭВМ 4-го поколения продолжается усложнение технических и программных структур (иерархия управления средствами, увеличение их количества). Следует отметить заметное повышение «интеллектуальности» машин. Особенно это стало видно при появлении персональных ЭВМ, ориентированных на определенные категории пользователей. Программное обеспечение этих машин создает дружественную среду общения человека и компьютера. Оно, с одной стороны, управляет процессом обработки информации, а с другой - создает необходимый сервис для пользователя, снижая трудоемкость его рутинной работы и предоставляя ему возможность больше внимания уделять творчеству.

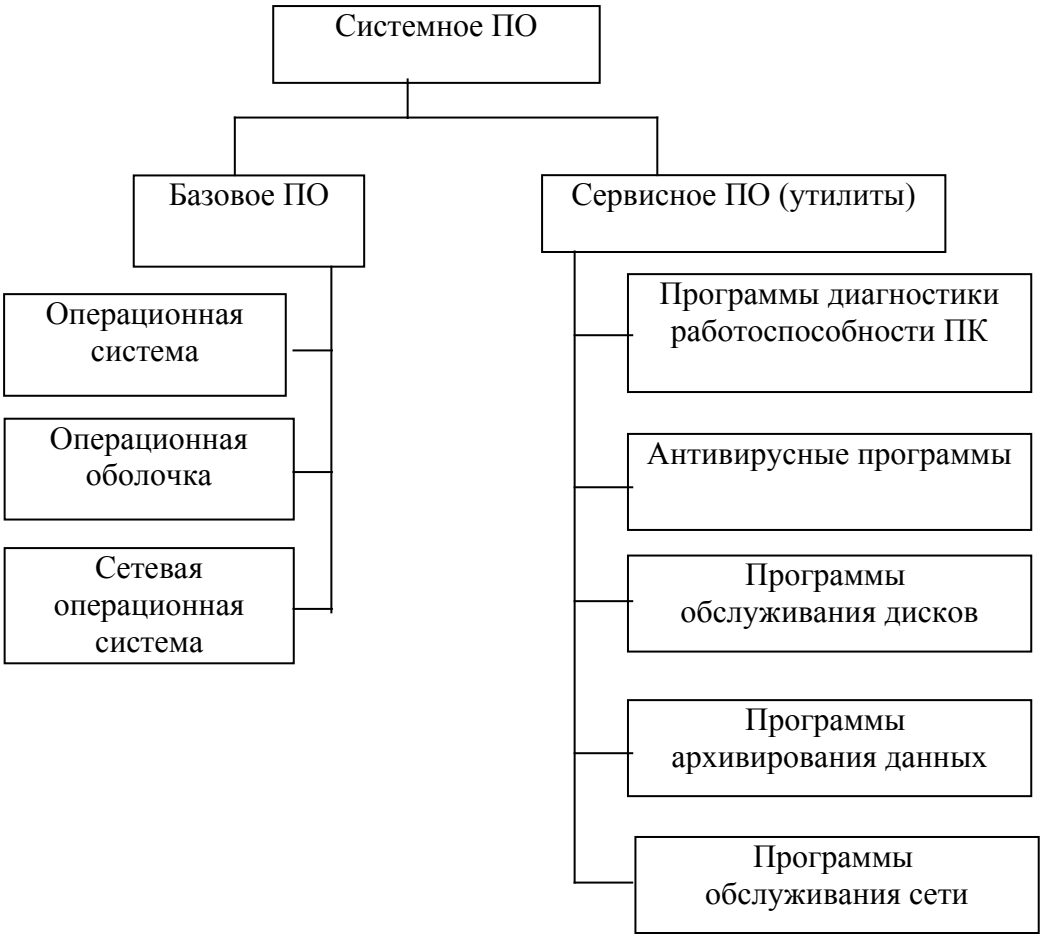
Подобные тенденции будут сохраняться и в ЭВМ следующих поколений. Так, по мнению исследователей, машины следующего столетия будут иметь встроенный в них искусственный интеллект, что позволит пользователям обращаться к машинам (системам) на естественном языке, вводить и обрабатывать тексты, документы, иллюстрации, создавать системы обработки знаний и т.д. Все это приводит к необходимости разработки сложного, иерархического программного обеспечения систем обработки данных.

Классификация программного обеспечения

Выделяют три основных класса программного обеспечения (ПО):

1. Системное программное обеспечение.
2. Пакеты прикладных программ (ППП).
3. Инструментарий технологии программирования (ИТП).

Системное программное обеспечение – совокупность программ и программных комплексов для обеспечения работы компьютера и сетей ЭВМ. К ПП данного класса применяются высокие требования по надежности и технологичности работы, удобству и эффективности использования.



Данный класс ПО ориентирован, в основном, на профессиональных пользователей: системного программиста, администратора сети, прикладного программиста, оператора. Однако и конечный пользователь должен знать базовую технологию работы с этим классом ПО.

Операционная система (ОС) – совокупность программных средств, обеспечивающая управление аппаратной частью компьютера и прикладными программами, а также их взаимодействие между собой и пользователем. ОС предназначена для управления выполнением пользовательских программ, планирования и управления вычислительными ресурсами ЭВМ. Доход от продаж ОС в среднем превышает 30 млрд. долларов в год (IBM, Microsoft, Novell, UNISYS).

ОС для персональных компьютеров делятся на:

одно– и многозадачные (в зависимости от числа параллельно выполняемых прикладных процессов);

одно– и многопользовательские (в зависимости от числа пользователей, одновременно работающих с ОС);

непереносимые и переносимые на другие типы компьютеров;

несетевые и сетевые, обеспечивающие работу в сетях ЭВМ.

ОС MS DOS фирмы Microsoft появилась в 1981г. В настоящее время существуют версии 6.22 и 7.0 (в составе Windows 95), а также ее разновидности других фирм-разработчиков (DR DOS, PC DOS). MS DOS – однозадачная, однопользовательская ОС. Обеспечивает командный интерфейс пользователя.

Операционные оболочки – специальные программы, предназначенные для облегчения общения пользователя с командами ОС. Операционные оболочки имеют текстовый и графический варианты интерфейса пользователя. Наиболее популярны текстовые оболочки для MS DOS: Norton Commander (Symantec); DOS Navigator. Эти программы существенно упрощают задание команды ОС для выполнения.

Большую популярность получили графические оболочки Windows 95 - Windows 98. Наиболее популярен следующий комплект утилит: Norton Utilities, Symantec. Наиболее известные антивирусные программы DrWeb, лаборатория Касперского, NOD32.

Пакет прикладных программ (ППП) – комплекс взаимосвязанных программ для решения задач определенного класса конкретного программного обеспечения.

В данный класс ПП входят пакеты, выполняющие обработку информации в различных предметных областях. Это самый многочисленный класс ПП.

Все ППП разделяются на:

- ✓ ППП общего назначения.
- ✓ Проблемно-ориентированные ППП (бухучет, отдел кадров, управление производством и т.д.);
- ✓ Системы автоматизированного проектирования САПР (для поддержки работы конструкторов и технологов, связанной с разработкой чертежей, схем и т.п.);
- ✓ Автоматизированные системы управления (АИС - автоматизированные информационные системы; АСНИ – автоматизированные системы научных исследований; АСРВ – автоматизированные системы реального времени; ИПС – информационно0поисковые системы; АСУ ТП (технологическими процессами); АСУП (производством));
- ✓ Методо-ориентированные ППП (математические, статистические и другие методы решения задач, независимо от сферы их применения);

- ✓ Офисные ППП (органайзеры, программы-переводчики, программы проверки орфографии, коммуникационные: электронная почта, браузеры для Internet и т.д.);
- ✓ Настольные издательские системы;
- ✓ Программные средства мультимедиа (процесс обучения, организация досуга);
- ✓ Системы искусственного интеллекта (экспертные системы, оболочки для их создания, системы управления базами знаний, системы анализа и распознавания речи и другие).

ППП общего назначения поддерживают информационные технологии конечных пользователей.

- ✓ настольные системы управления базами данных (СУБД);
- ✓ серверы БД (для создания и работы в сети интегрированных БД);
- ✓ генераторы отчетов (серверы) – реализация запросов и генерация отчетов в условиях сети с архитектурой клиент-сервер;
- ✓ текстовые и табличные процессоры;
- ✓ средства презентационной графики – для подготовки слайд-фильмов, мультфильмов, видеофильмов, их редактирования;
- ✓ интегрированные пакеты – набор нескольких ПП, функционально дополняющие друг друга и поддерживающие единые информационные технологии.

Инструментальные средства разработки ПО – совокупность программ и программных комплексов, обеспечивающих технологию разработки, отладки и внедрения создаваемых ПП. Включают языки программирования, системы программирования, инструментальные среды пользователя (средства тестирования и отладки), инструментальные средства связи с объектом (УСО).

Основные характеристики программ:

1. Алгоритмическая сложность.
2. Состав и глубина проработки реализованных функций.
3. Полнота и системность функций.
4. Объем файлов программ.
5. Требования к операционной системе и техническим средствам со стороны программы.
6. Объем дисковой памяти.

7. Размер оперативной памяти.

8. Тип процессора.

9. Версия операционной .системы.

10. Наличие вычислительной сети и др.

Показатели качества программного продукта (ПП)

Номенклатура и требуемые значения показателей качества определяются прежде всего *функциональным назначением* конкретных ПП. Прежде всего, хорошая программа должна делать то, что ожидает от нее заказчик – то есть удовлетворять требованиям заказчика. Это приводит к широкому спектру показателей качества в спецификации требований к программному продукту.

Про факторы качества и цели программирования много писалось, а также много делалось попыток их выделения и анализа. В общем случае под ***качеством (quality) программ*** понимается то, насколько они соответствуют установленным для них требованиям - спецификациям и сколько высоко установлена планка этих требований, это совокупность черт и характеристик ПП, которые влияют на ее способность удовлетворять заданные потребности пользователей.

1. ***Надежность (reliability)*** - это способность системы ПО выполнять возложенные на нее функции при поступлении требований на их выполнение. Понятие надежности существенно отличается от понятия доказательства правильности программы, т.к. *правильность* - это некоторое *статическое свойство*, а надежность относится к *динамическим требованиям*, предъявляемым к системе, и способности системы удовлетворять этим требованиям. Надежное ПО обеспечивает бесперебойность и устойчивость в работе, дает возможность диагностики возникающих ошибок.

Надежность ПО включает в себя такие составные свойства, как:

- *отказоустойчивость* – возможность восстановления программы и данных в случае сбоев в работе;
- *безопасность* – сбои в работе программы не должны приводить к опасным последствиям (авариям);
- *защищенность* от случайных или преднамеренных внешних воздействий («защита от дурака», вирусов, спама т.п.).

2. ***Мобильность*** – независимость ПП от технических средств обработки информации, ОС, сетевой технологии. Мобильный ПП пригоден для массового использования без каких-либо изменений. Машины и технические средства развиваются и

дешевеют быстрее, чем программы. Поэтому ПО должно быть легко переносимым на новые и более дешевые машины и с одной платформы на другую.

3. ***Эффективность*** – это отношение уровня услуг, предоставляемых ПС пользователю при заданных условиях, к объему используемых ресурсов. Эффективность ПО оценивается следующими показателями: время выполнения кода, загруженность процессора, объем требуемой памяти, время отклика и т.п. (минимально возможный расход вычислительных ресурсов и максимально возможное быстродействие).

4. ***Модифицируемость*** - простота внесения изменений.

5. ***Удобство использования (usability).*** ПП должен быть легким в использовании, причем именно тем типом пользователей, на которых он рассчитан. Это включает в себя интерфейс пользователя и адекватную документацию. Причем, пользовательский интерфейс должен быть не интуитивно, а профессионально понятным пользователю.

6. ***Коммуникативность*** – свойство интеграции с другими программами, обеспечения обмена данными в общих форматах представления.

7. ***Учет человеческого фактора*** – обеспечение дружественного интерфейса, наличие контекстно-зависимой подсказки или обучающей системы, хорошей документации.

ГЛАВА 2 ИНФОРМАЦИОННО-ЛОГИЧЕСКИЕ ОСНОВЫ ЭВМ

2.1. Системы счисления

Системой счисления называется способ изображения чисел с помощью ограниченного набора символов, имеющих определенные количественные значения. Систему счисления образует совокупность правил и приемов представления чисел с помощью набора знаков (цифр).

Различают позиционные и непозиционные системы счисления. В позиционных системах каждая цифра числа имеет определенный вес, зависящий от позиции цифры в последовательности, изображающей число. Позиция цифры называется разрядом. В позиционной системе счисления любое число можно представить в виде:

$$A_n=a_{m-1}a_{m-2}\ldots a_i a_0 * a_{-1}a_{-2}\ldots a_{-k}=a_{m-1}*N^{m-1}+a_{m-2}*N^{m-2}\ldots+a_{-k}*N^{-k}, \quad (2.1)$$

где a_i – i-я цифра числа; k – количество цифр в дробной части числа; m - количество цифр в целой части числа; N – основание системы счисления.

Основание системы счисления N показывает, во сколько раз «вес» i-го разряда больше (i-1) разряда. Целая часть числа отделяется от дробной части точкой (запятой).

Пример 2.1.

$A_{10}=37.25$.

В соответствии с формулой (2.1) это число формируется из цифр с весами рядов:

$A_{10}=3*10^1+7*10^0+2*10^{-1}+5*10^{-2}$.

Теоретически наиболее экономичной системой счисления является система с основанием e=2,71828..., находящимся между числами 2 и 3.

Во всех современных ЭВМ для представления числовой информации используется двоичная система счисления. Это обусловлено:

- ✓ более простой реализацией алгоритмов выполнения арифметических и логических операций;
- ✓ более надежной физической реализацией основных функций, так как они имеют всего два состояния (0 и 1);
- ✓ экономичностью аппаратурной реализации всех схем ЭВМ.

При N=2 число различных цифр, используемых для записи чисел, ограничено множеством из двух цифр (нуль и единица). Кроме двоичной системы счисления широкое распространение получили и производные системы:

- *десятичная*, точнее двоично-десятичное представление десятичных чисел, - {0, 1,...,9};
- *шестнадцатеричная* - {0,1,2, ...9, A, B, C, D, E, F}. Здесь шестнадцатеричная цифра A обозначает число 10,B-число 11, ...,F-число 15;
- *восьмеричная* (от слова *восьмерик)* - {0,1,2,3,4,5, б, 7}. Она широко используется во многих специализированных ЭВМ.

Восьмеричная и шестнадцатеричная системы счисления являются производными от двоичной, так как $16 = 2^4$ и $8 = 2^3$. Они используются в основном для более компактного изображения двоичной информации, так как запись значения чисел производится существенно меньшим числом знаков.

Пример 2.2.

Число в двоичной, восьмеричной и шестнадцатеричной системах счисления имеет следующее представление:

A_2=1100100,101;

A_g=144.5;

A_{16}=64.A;

$A_2=1*2^6+1*2^5+0*2^4+0*2^3+1*2^2+0*2^1+1*2^0+1*2^{-1}+0*2^{-2}+1*2^{-3}$;

$A_8=1*8^2+4*8^1+4*8^0+5*8^{-1}$;

$A_{16}=6*16^1+4*16^0+10*16^{-1}$.

Представление чисел в различных системах счисления допускает однозначное преобразование их из одной системы в другую. В ЭВМ перевод из одной системы в другую осуществляется автоматически по специальным программам. Правила перевода целых и дробных чисел отличаются.

2.2.1. Представление числовой информации

В ЭВМ используются три вида чисел:

- с фиксированной точкой (запятой),
- с плавающей точкой (запятой);
- двоично-десятичное представление.

Точка (запятая) - это подразумеваемая граница целой и дробной частей числа.

У *чисел с фиксированной точкой* в двоичном формате предполагается строго определенное место точки (запятой). Обычно это место определяется или перед первой значащей цифрой числа, или после последней значащей цифрой числа. Если точка

фиксируется перед первой значащей цифрой, то это означает, что число по модулю меньше единицы. Диапазон изменения значений чисел определяется неравенством

$$2^{-n} \le |A_2| \le 1 - 2^{-n}.$$

Если точка фиксируется после последней значащей цифры, то это означает, что n-разрядные двоичные числа являются целыми. Диапазон изменения их значений составляет:

$$0 \le |A^2| \le 2^n - 1.$$

Перед самым старшим из возможных разрядов двоичного числа фиксируется его знак. Положительные числа имеют нулевое значение знакового разряда, отрицательные - единичные.

Другой формой представления чисел является представление их в виде чисел *с плавающей точкой (запятой).* Числа с плавающей точкой представляются в виде мантиссы n_a и порядка p_a , иногда это представление называют полулогарифмической формой числа. Например, число A_{10}= 373 можно представить в виде $0.373 \cdot 10^3$, при этом n_a = 0.373, p = 3, основание системы счисления подразумевается фиксированным и равным десяти. Для двоичных чисел A_2 в этом представлении также формируется n_a и порядок p_a при основании системы счисления равным двум.

$$0 \le |A^2| \le 2^n - 1.$$

что соответствует записи

$$p_a^{max} = 2^r - 1$$

Порядок числа p_a определяет положение точки (запятой) в двоичном числе. Значение порядка лежит в диапазоне $-p_{amax} <= p_a <= p_{amax}$, где величина p_{amax} определяется числом разрядов r, отведенных для представления порядка

$$p_a^{max} = 2^r - 1$$

Положительные и отрицательные значения порядка значительно усложняют обработку вещественных чисел. Поэтому во многих современных ЭВМ используют не прямое значение p_a, а модифицированное p'_a приведенное к интервалу

$$0 \le p'_a \le 2p_a^{max}$$

Значение p'_a носит название «характеристика числа». Обычно под порядок (модифицированный порядок - характеристику) выделяют один байт. Старший разряд

характеристики отводится под знак числа, а семь оставшихся разрядов обеспечивают изменение порядка в диапазоне

$$-64 \le P_a \le 63$$

Модифицированный порядок p'_a вычисляется по зависимости

$$p'_a = p_a + 64$$

Этим самым значения p'_a формируются в диапазоне положительных чисел

$$0 \le |m_a| \le 1 - 2^{-k}$$

Мантисса числа m_a представляется двоичным числом, у которого точка фиксируется перед старшим разрядом, т. е.

$$0 \le |m_a| \le 1 - 2^{-k}$$

где k - число разрядов, отведенных для представления мантиссы.

Если $1/N \le |m_a| \le 1 - 2^{-k}$, то старший значащий разряд мантиссы в системе счисления с основанием N отличен от нуля. Такое число называется ***нормализованным***. Например, $A_2 = (100;0.101101)_2$ - нормализованное число $A_2 = 1011.01$ или $A_{10} = 11.25$, а то же самое число $A_2 = (101 ;0.0101101)$ - число ненормализованное, так как старший разряд мантиссы равен нулю.

Диапазон представления нормализованных чисел с плавающей точкой определяется

$$2^{-1} \cdot 2^{-(2^r-1)} \le |A_2| \le (1 - 2^{-k}) \cdot 2^{(2^r-1)}$$

где r и k - соответственно количество разрядов, используемых для представления порядка и мантиссы.

Третья форма представления двоичных чисел - *двоично-десятичная*. Ее появление объясняется следующим. При обработке больших массивов десятичных чисел (например, больших экономических документов) приходится тратить существенное время на перевод этих чисел из десятичной системы счисления в двоичную для последующей обработки и обратно - для вывода результатов. Каждый такой перевод требует выполнения двух - четырех десятков машинных команд. С включением в состав отдельных ЭВМ специальных функциональных блоков или спецпроцессоров десятичной арифметики появляется возможность обрабатывать десятичные числа напрямую, без их преобразования, что сокращает время вычислений. При этом каждая цифра десятичного числа представляется двоичной тетрадой. Например, $A_{10}=3759$, $A_{2\text{-}10}= 0011\ 0111\ 0101\ 1001$. Положение десятичной точки (запятой), отделяющей целую часть от дробной, обычно

заранее фиксируется. Значение знака числа отмечается кодом, отличным от кодов цифр. Например, «+» имеет значение тетрады «1100», а «-» - «1101».

2.2.2. Представление других видов информации

Различные виды информации могут быть разделены на две группы: *статические* и *динамические*. Так, числовая, логическая и символьная информация является статической – ее значение не связано со временем. В отличие от перечисленных типов вся аудиоинформация имеет динамический характер. Она существует только в режиме реального времени, ее нельзя остановить для более подробного изучения. Если изменить масштаб времени (увеличить или уменьшить), аудиоинформация искажается. Это свойство иногда используется для получения звуковых эффектов.

Видеоинформация может быть как статической, так и динамической. Статическая видеоинформация включает текст, рисунки, графики, чертежи, таблицы и др. Рисунки делятся также на плоские - двухмерные и объемные - трехмерные.

Динамическая видеоинформация - это видео-, мульт- и слайд- фильмы. В их основе лежит последовательное экспонирование на экране в реальном масштабе времени отдельных кадров в соответствии со сценарием.

Динамическая видеоинформация используется либо для передачи движущихся изображений (анимация), либо для последовательной демонстрации отдельных кадров вывода (слайд-фильмы).

Для демонстрации анимационных и слайд-фильмов используются различные принципы. Анимационные фильмы демонстрируются так, чтобы зрительный аппарат человека не мог зафиксировать отдельных кадров. В современных высококачественных мониторах и в телевизорах с цифровым управлением электронно-лучевой трубкой кадры сменяются до 70 раз в секунду, что позволяет высококачественно передавать движение объектов.

При демонстрации слайд-фильмов каждый кадр экспонируется на экране столько времени, сколько необходимо для восприятия его человеком (обычно от 30 с до 1 мин). Слайд-фильмы можно отнести к статической видеоинформации.

По способу формирования видеоизображения бывают:

- растровые,
- матричные
- векторные.

Растровые видеоизображения используются в телевидении, а в ЭВМ практически не применяются.

Матричные изображения получили в ЭВМ наиболее широкое распространение. Изображение на экране рисуется электронным лучом точками.

Информация представляется в виде характеристик значений каждой точки - пиксела (picture element), рассматриваемого как наименьшей структурной единицей изображения. Количество высвечиваемых одновременно пикселов на экране дисплея определяется его разрешающей способностью. В качестве характеристик графической информации выступают: координаты точки (пиксела) на экране, цвет пиксела, цвет фона (градация яркости). Вся эта информация хранится в видеопамяти дисплея. При выводе графической информации на печать изображение также воспроизводится по точкам.

Изображение может быть и в ***векторной*** форме. Тогда оно составляется из отрезков линий (в простейшем случае - прямых), для которых задаются: начальные координаты, угол наклона и длина отрезка (может указываться и код используемой линии). Векторный способ имеет ряд преимуществ перед матричным: изображение легко масштабируется с сохранением формы, является “прозрачным” может быть наложено на любой фон и т.д.

Способы представления информации в ЭВМ, кодирование и преобразование кодов в значительной степени зависят от принципа действия устройств, в которых эта информация формируется, накапливается, обрабатывается и отображается.

Для кодирования символьной или текстовой информации применяются различные системы: при вводе информации с клавиатуры кодирование происходит при нажатии клавиши, на которой изображен требуемый символ, при этом в клавиатуре вырабатывается так называемый scan-код, представляющий собой двоичное число, равное порядковому номеру клавиши.

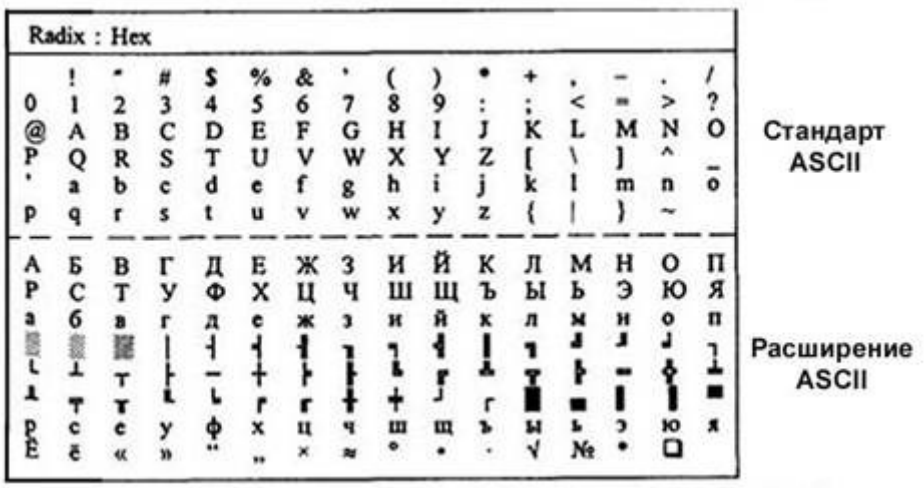
Номер нажатой клавиши никак не связан с формой символа, нанесенного на клавише. Опознание символа и присвоение ему внутреннего кода ЭВМ производятся специальной программой по специальным таблицам: ДКОИ, КОИ-7, ASCII (Американский стандартный код передачи информации).

Всего с помощью таблицы кодирования ASCII (таблица 2.1) можно закодировать 256 различных символов. Эта таблица разделена на две части: основную (с кодами от 00_h до $7F_h$) и дополнительную (от 80_h до FF_h, где буква h обозначает принадлежность кода к шестнадцатеричной системе счисления).

Первая половина таблицы стандартизована. Она содержит управляющие коды (от 00_h до 20_h и 77_h). Эти коды из таблицы изъяты, так как они не относятся к текстовым элементам. Здесь же размещаются знаки пунктуации и математические знаки: 21_h - !, 26_h - &, 28_h - (, $2B_h$ -+,..., большие и малые латинские буквы: 41_h - A, 61_h - a,...

Вторая половина таблицы содержит национальные шрифты, символы псевдографики, из которых могут быть построены таблицы, специальные математические знаки. Нижнюю часть таблицы кодировок можно заменять, используя соответствующие *драйверы* - управляющие вспомогательные программы. Этот прием позволяет применять несколько шрифтов и их гарнитур.

Таблица 2.1 – Таблица кодирования текстовой информации ASCII



Дисплей по каждому коду символа должен вывести на экран изображение символа - не просто цифровой код, а соответствующую ему картинку, так как каждый символ имеет свою форму. Описание формы каждого символа хранится в специальной памяти дисплея - знакогенераторе. Высвечивание символа на экране дисплея IBM PC осуществляется с помощью точек, образующих символьную матрицу. Каждый пиксел в такой матрице является элементом изображения и может быть ярким или темным. Темная точка кодируется цифрой 0, светлая (яркая)- 1. Если изображать в матричном поле знака темные пикселы точкой, а светлые - звездочкой, то можно графически изобразить форму символа.

Кодирование аудиоинформации – процесс более сложный. Аудиоинформация является аналоговой. Для преобразования ее в цифровую форму используют аппаратурные средства: аналого-цифровые преобразователи (АЦП), в результате работы которых аналоговый сигнал оцифровывается – представляется в виде числовой последовательности. Для вывода оцифрованного звука на аудиоустройства необходимо проводить обратное преобразование, которое осуществляется с помощью цифро-аналоговых преобразователей (ЦАП).

2.3. Арифметические основы ЭВМ

Все современные ЭВМ имеют достаточно развитую систему команд, включающую десятки и сотни машинных операций. Однако выполнение любой операции основано на использовании простейших микроопераций типа сложения и сдвиг. Это позволяет иметь

единое арифметико-логическое устройство для выполнения любых операций, связанных с обработкой информации. Правила сложения двоичных цифр двух чисел А и В представлены в таблице 2.2.

Здесь показаны правила сложения двоичных цифр a_i, b_i одноименных разрядов с учетом возможных переносов из предыдущего разряда p_{i-1}.

Таблица 2.2 – Правила сложения двоичных цифр

Значения двоичных чисел А и В			Разряд суммы S_i	Перенос в следующий разряд P_i
a_i	b_i	P_{i-1}		
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

Подобные таблицы можно было бы построить для любой другой арифметической и логической операции (вычитание, умножение и т.д.), но именно данные этой таблицы положены в основу выполнения любой операции ЭВМ. Под знак чисел отводится специальный знаковый разряд. Знак «+» кодируется двоичным нулем, а знак «-» - единицей. Действия над прямыми кодами двоичных чисел при выполнении операций создают большие трудности, связанные с необходимостью учета значений знаковых разрядов:

- ✓ во-первых, следует отдельно обрабатывать значащие разряды чисел и разряды знака;
- ✓ во-вторых, значение разряда знака влияет на алгоритм выполнения операции (сложение может заменяться вычитанием и наоборот).

Во всех ЭВМ без исключения все операции выполняются над числами, представленными специальными машинными кодами. Их использование позволяет обрабатывать знаковые разряды чисел так же, как и значащие разряды, а также заменять операцию вычитания операцией сложения.

Различают ***прямой код*** (П), ***обратный код*** (ОК) и ***дополнительный код*** (ДК) двоичных чисел.

2.3.1. Машинные коды

Прямой код двоичного числа образуется из абсолютного значения этого числа и кода знака (нуль или единица) перед его старшим числовым разрядом.

Пример 2.3.

A_{10}=+10 A_2=+1010 $[A_2]_П$=0:1010;

B_{10}=-15 B_2=-1111 $[B_2]_П$=1:1111.

Точечной вертикальной линией здесь отмечена условная граница, отделяющая знаковый разряд от значащих.

Обратный код двоичного числа образуется по следующему правилу. Обратный код положительных чисел совпадает с их прямым кодом. Обратный код отрицательного числа содержит единицу в знаковом разряде числа, а значащие разряды числа заменяются на инверсные, т.е. нули заменяются единицами, а единицы - нулями.

Пример 2.4.

A_{10}=+5 A_2=+101 $[A_2]_П$=$[A_2]_{ОК}$=0:101;

B_{10}=-13 B_2=-1010 $[B_2]_{ОК}$=1:0010.

Свое название обратный код чисел получил потому, что коды цифр отрицательного числа заменены на инверсные. Укажем наиболее *важные свойства обратного кода* чисел:

• сложение положительного числа с его отрицательным значением в обратном коде дает так называемую машинную единицу МЕок= 1: 111... 11, состоящую из единиц в знаковом и значащих разрядах числа;

• нуль в обратном коде имеет двоякое значение. Он может быть положительным - 0: 00...0 и отрицательным числом - 1; 11... 11. Значение отрицательного нуля совпадает с МЕок. Двойственное представление нуля явилось причиной того, что в современных ЭВМ все числа представляются не обратным, а дополнительным кодом. Дополнительный код положительных чисел совпадает с их прямым кодом. Дополнительный код отрицательного числа представляет собой результат суммирования обратного кода числа с единицей младшего разряда (2° - для целых чисел, 2-k - для дробных).

Пример 2.5.

A_{10}=+19 A_2=+10011 $[A_2]_П$=$[A_2]_{ОК}$=$[A_2]_{ДК}$=0:10011;

B_{10}=-13 B_2=-1101 $[B2]_{ДК}$=$[B_2]_{ОК}$+2_0=1:0010+1=1:0011.

Укажем основные *свойства дополнительного кода*:

• сложение дополнительных кодов положительного числа С с его отрицательным значением дает так называемую ***машинную единицу дополнительного кода***:

$МЕ_{ДК}=МЕ_{ОК}+2^0=10: 00…00,$

т.е. число 10 (два) в знаковых разрядах числа;

• дополнительный код получил такое свое название потому, что представление отрицательных чисел является дополнением прямого кода чисел до машинной единицы МЕдк.

Модифицированные обратные и *дополнительные коды* двоичных чисел отличаются соответственно от обратных и дополнительных кодов удвоением значений знаковых разрядов. Знак «+» в этих кодах кодируется двумя нулевыми знаковыми разрядами, а «-» - двумя единичными разрядами.

Пример 2.6.

$A_{10}=9$ $A_2=+1001$ $[A2]_П=[A2]_{ОК}=[A2]_{ДК}=0:1001$

$[A_2]_{МОК}=[A_2]_{МДК}=00:1001;$

$B_{10}=-9$ $B_2=-1001$ $[B_2]_{ОК}=1:0110$ $[B_2]_{ДК}=1:0111$

$[B_2]_{МОК}=11:0110$ $[B_2]_{МДК}=11:0111.$

Целью введения модифицированных кодов являются фиксация и обнаружение случаев получения неправильного результата, когда значение результата превышает максимально возможный результат в отведенной разрядной сетке машины. В этом случае перенос из значащего разряда может исказить значение младшего знакового разряда. Значение знаковых разрядов “01” свидетельствует о положительном переполнении разрядной сетки, а “10” - об отрицательном переполнении. В настоящее время практически во всех моделях ЭВМ роль удвоенных разрядов для фиксации переполнения разрядной сетки играют переносы, идущие в знаковый и из знакового разряда.

Сложение (вычитание)

Операция вычитания приводится к операции сложения путем преобразования чисел в обратный или дополнительный код. Пусть числа $A \geq O$ и $B \geq O$, тогда операция алгебраического сложения выполняется в соответствии с таблице 2.3.

Таблица 2.3 -Таблица преобразования кодов при алгебраическом сложении

Требуемая операция	Необходимое преобразование
A+B	A+B
A-B	A+(-B)
-A+B	(-A)+B
-A-B	(-A)+(-B)

Скобки в представленных выражениях указывают на замену операции вычитания операцией сложения с обратным или дополнительным кодом соответствующего числа. Сложение двоичных чисел осуществляется последовательно, поразрядно в соответствии с таблицей 2.2. При выполнении сложения цифр необходимо соблюдать следующие правила:

- *Слагаемые должны иметь одинаковое число разрядов*. Для выравнивания разрядной сетки слагаемых можно дописывать незначащие нули слева к целой части числа и незначащие нули справа к дробной части числа.
- *Знаковые разряды* чисел участвуют в сложении так же, как и значащие.

Необходимые преобразования кодов (п. 2.3.1) производятся с изменением знаков чисел. Приписанные незначащие нули изменяют свое значение при преобразованиях по общему правилу. При образовании единицы переноса из старшего знакового разряда, в случае использования **ОК**, эта единица складывается с младшим числовым разрядом. При использовании **ДК** единица переноса теряется. Знак результата формируется автоматически, результат представляется в том коде, в котором представлены исходные слагаемые.

Пример 2.7. Сложить два числа $A_{10}=7$ $B_{10}=16$

A_2=+11=+0111;

B_2=+1000=+10000.

Исходные числа имеют различную разрядность, необходимо провести выравнивание разрядной сетки:

$[A_2]_{П}=[A_2]_{ОК}=[A_2]_{ДК}$=0: 00111;

$[B_2]_{П}=[B_2]_{ОК}=[B_2]_{ДК}$=0: 10000.

Сложение в обратном или дополнительном коде дает один и тот же результат

$$\begin{array}{r} 0\vdots\ 00111 \\ +0\vdots\ 10000 \\ \hline C_2 = 0\vdots\ 10111 \end{array}$$

$C_{10} = +23.$

Обратите внимание, что при сложении цифр отсутствуют переносы в знаковый разряд и из знакового разряда, что свидетельствует о получении правильного результата.

Пример 2.8. Сложить два числа $A_{10} = +16$ $B_{10} = -7$ в ОК и ДК. В соответствии с таблицей 2.3 должна быть реализована зависимость A+(-B), в которой второй член преобразуется с учетом знака

$[A_2]_п = 0\vdots\ 10000 = 0\vdots\ 10000$ $[A_2]_{ок} = 0\vdots\ 10000$ $[A_2]_{дк} = 0\vdots\ 10000;$

$[B_2]_п = 1\vdots\ 111 = 1\vdots\ 00111$ $[B_2]_{ок} = 1\vdots\ 11000$ $[B_2]_{дк} = 1\vdots\ 11001.$

Сложение в ОК

$$\begin{array}{r} [A_2]_{ок} = 0\vdots\ 10000 \\ +[B_2]_{ок} = 1\vdots\ 11000 \\ \hline 0\vdots\ 01000 \\ +\quad\quad\quad 1 \\ \hline 0\vdots\ 01001 \end{array}$$

$C_2 = 0\vdots\ 01001$

$C_{10} = +9$

Сложение в ДК

$$\begin{array}{r} [A_2]_{дк} = 0\vdots\ 10000 \\ +[B_2]_{дк} = 1\vdots\ 11001 \\ \hline 0\vdots\ 01001 \end{array}$$

$C_2 = 0\vdots\ 01001$

$C_{10} = +9$

При сложении чисел в ОК и ДК были получены переносы в знаковый разряд и из знакового разряда. В случае ОК перенос из знакового разряда требует дополнительного прибавления единицы младшего разряда (см.п.4 правил). В случае ДК этот перенос игнорируется.

Умножение

Умножение двоичных чисел наиболее просто реализуется в прямом коде. Рассмотрим, каким образом оно приводится к операциям сложения и сдвигам.

Пример 2.9. Умножить два числа A_{10}=7 B_{10}=5.

Перемножим эти числа, представленные прямыми двоичными кодами, так же, как это делается в десятичной системе.

$[A_2]_П = 0 \mid 111$ $[B2]_П = 0 \mid 101$

0 | 111

0 | 101

111

111 .

$[C_2]_П = 100011$

Нетрудно видеть, что произведение получается путём сложения частных произведений, представляющих собой разряды множимого, *сдвинутые влево* в соответствии с позициями разрядов множителя. Частные произведения, полученные умножением на нуль, игнорируются. Важной особенностью операции умножения n-разрядных сомножителей является увеличение разрядности произведения до n+n=2n. Знак произведения формируется путём сложения знаковых разрядов сомножителей. Возможные переносы из знакового разряда игнорируются.

Деление

Операция деления, как и в десятичной арифметике, является обратной операции умножения. Покажем, что и эта операция приводится к последовательности операций сложения и сдвига.

Пример 2.10. Разделить два числа $A_{10}=45$ $B_{10}=5$

$[A_2]_П = 101101$
$[B_2]_П = 101$

Делимое	Делитель
101101	| 101
-101	1001 – частное
0101	
- 101	
0	

$[C_2]_П = 1001$
$C_{10} = 9$

Деление произведено так же, как это делается обычно в десятичной системе. Сначала проверяется, можно ли вычесть значение делителя из старших разрядов делимого. Если возможно, то в разряде частного записывается единица и определяется частная разница. В противном случае в частное записывается нуль и разряды делителя сдвигаются вправо на один разряд по отношению к разрядам делимого. К полученной предыдущей разнице сносится очередная цифра делимого, и данный процесс повторяется, пока не будет получена необходимая точность. Если учесть, что все вычитания в ЭВМ заменяются сложением в ОК или в ДК (см. таблицу 2.3), то действительно операция

деления приводится к операциям сложения и сдвигам вправо разрядов делителя относительно разрядов делимого.

Замечание1: делимое перед операцией деления должно быть приведено к $2n$-разрядной сетке. Только в этом случае при делении на n-разрядный делитель получается n-разрядное частное.

Замечание 2: Знак частного формируется также путем сложения знаковых разрядов делимого и делителя, как это делалось при умножении.

2.3.2. Арифметические операции над двоичными числами с плавающей точкой

В современных ЭВМ **числа с плавающей точкой** *хранятся* в памяти машин, имея мантиссу и порядок (характеристику) ***в прямом коде и нормализованном виде***. Все арифметические действия над этими числами выполняются так же, как это делается с ними, если они представлены в полулогарифмической форме (мантисса и десятичный порядок) в десятичной системе счисления. Порядки и мантиссы обрабатываются раздельно.

Сложение (вычитание). Операция сложения (вычитания) производится в следующей последовательности.

1. Сравниваются порядки (характеристики) исходных чисел путем их вычитания $p=p_1-p_2$. При выполнении этой операции определяется, одинаковый ли порядок имеют исходные слагаемые.
2. Если разность порядков равна нулю, то это значит, что одноименные разряды мантисс имеют одинаковые веса (двоичный порядок). В противном случае должно проводиться выравнивание порядков.
3. Для выравнивания порядков число с меньшим порядком сдвигается вправо на разницу порядков Ap. Младшие выталкиваемые разряды при этом теряются.
4. После выравнивания порядков мантиссы чисел можно складывать (вычитать) в зависимости от требуемой операции. Операция вычитания заменяется операцией сложения в соответствии с данными таблицы 2.3. Действия над слагаемыми производятся в ОК или ДК по общим правилам.
5. Порядок результата берется равным большему порядку.
6. Если мантисса результата не нормализована, то осуществляются нормализация и коррекция значений порядка.

Пример 2.11. Сложить два числа $A_{10}=+1.375$; $B_{10}=-0.625$.

A_2=+1.011=0: 1011*101; B_2=-0.101=-0:101*100.

В нормализованном виде эти числа будут иметь вид:

$$\begin{array}{ll} p_1 = 0{:}1 & [p_1]_{дк} = 0{:}1 \\ p_2 = 0{:}0 & + \quad [p_2]_{дк} = 0{:}0 \\ & \overline{\Delta p = 0{:}1} \end{array}$$

1. Вычитаем порядки Δp=p1-p2=1-0=1. В машине эта операция требует операции сложения с преобразованием порядка чисел в дополнительный код:

$$\begin{array}{ll} Порядок & Мантиса \\ [A_2]_П = 0{:}1 & \uparrow 0{:}1011 \\ & знак\ числа \\ [B_2]_П = 0{:}0 & \downarrow 1{:}101 \end{array}$$

Определяем, что Δp≠ 0.

2. Порядок первого числа больше порядка второго числа на единицу. Требуется выравнивание порядков.

3. Для выравнивания порядков необходимо второе число сдвинуть вправо на один разряд.

$[B_2]_{исх}$=0: 0 1: 101

после сдвига

$[B_2]_{п}$=0: 11:0101

$[m_B]_{дк}$= 1: 1011

4. Складываем мантиссы.

$$\begin{array}{l} [m_A]_{дк} = 0{:}1011 \\ +[m'_B]_{дк} = 1{:}1011 \\ \overline{[m_C]_{дк} = 0{:}0110} \end{array}$$

Мантисса числа С - положительная.

5. Порядок числа С равен порядку числа с большим порядком, т.е. p = +1.

[C2]п=0: 1 0: 0110.

Видно, что мантисса результата не нормализована, так как старшая цифра мантиссы равна нулю.

6. Нормализуем результат путем сдвига мантиссы на один разряд влево и соответственно вычитаем из значения порядка единицу:

$$[C_2]_{\text{П}} = 0{:}00{:}110,$$
$$C_{10} = +0.75.$$

Умножение (деление). Операция умножения (деления) чисел с плавающей точкой также требует разных действий над порядками и мантиссами. Алгоритмы этих операций выполняются в следующей последовательности.

1. При умножении (делении) порядки складываются (вычитаются) так, как это делается над числами с фиксированной точкой.
2. При умножении (делении) мантиссы перемножаются (делятся).
3. Знаки произведения (частного) формируются путем сложения знаковых разрядов сомножителей (делимого и делителя). Возможные переносы из знакового разряда игнорируются.

2.3.4. Арифметические операции над двоично-десятичными кодами чисел

При обработке больших массивов экономической информации переводы чисел из десятичной системы в двоичную и обратно могут требовать значительного машинного времени. Некоторые образцы ЭВМ поэтому имеют или встроенные, или подключаемые блоки, которые обрабатывают десятичные целые числа в их двоично-десятичном представлении. Действия над ними также приводятся к операции алгебраического сложения отдельных цифр чисел, представленных дополнительными кодами в соответствии с таблице 2.3.

Приведем один из *алгоритмов сложения*, который получил довольно широкое распространение.

1. Сложение чисел начинается с младших цифр (тетрад) и производится с учетом возникающих переносов из младших разрядов в старшие.
2. Знак суммы формируется специальной логической схемой по знаку большего слагаемого.
3. Для того чтобы при сложении двоично-десятичных цифр возникали переносы, аналогичные при сложении чисел в десятичном представлении, необходимо проводить так называемую *десятичную коррекцию*. Для этого к каждой тетраде первого числа прибавляется дополнительно по цифре $6_{10}=0110_2$, что позволяет исключить шесть неиспользуемых комбинаций $(1010\text{-}1111)_2$, так как они кодируют шестнадцатеричные цифры A-F (числа $10\text{-}15_{10}$).
4. После операции суммирования осуществляется корректировка суммы. Из тех тетрад суммы, из которых не было переносов, изымаются ранее внесенные избытки $6_{10}=0110_2$. Для этого проводится вторая коррекция. Операция вычитания заменяется,

как и обычно, операцией сложения с числом -6, представленным дополнительным кодом 1010_2, но только в тех разрядах, в которых отсутствовали переносы. При этой второй коррекции переносы из тетрад блокируются.

5. Операция вычитания реализуется достаточно своеобразно. По общему правилу сложения (п.п.1-4) к тетрадам числа с большим модулем прибавляются дополнительные коды тетрад другого числа. В качестве знаке результата берется знак числа с большим модулем.

Пример 2.14. Сложить два числа $A_{10}=177$; $B_{10}=418$

A_{2-10}	00001	0111	0111	
+				1-я коррекция
	0110	0110	0110	
A'	0111	1101	1101	
+				Сложение $A'+B$
B_{2-10}	0100	0001	1000	
	1011	1111	0101	результат C'
+				
	1010	1010		2-я коррекция
C_{2-10}	0101	1001	0101	результат

$C_{10}=595$

2.4 Логические основы ЭВМ

2.4.1.Основные сведения из алгебры логики

Теоретической основой построения ЭВМ являются специальные математические дисциплины. Одной из них является ***алгебра логика*** или булева алгебра.[*] Ее аппарат широко используют для описания схем ЭВМ, их оптимизации и проектирования.

Вся информация в ЭВМ представляется в двоичной системе счисления. Поставим в соответствие входным сигналам отдельных устройств ЭВМ соответствующие значения xi(i=1,n), а выходным сигналам - значения функций y_j (j=1,m) (рисунок 2.1).

Рисунок 2.1 - Представление схемы ЭВМ

В этом случае зависимостями

$$y_j = f(x_1, x_2, \ldots, x_i, \ldots, x_n), \qquad (2.2)$$

где x_i – i-й вход; n – число входов; y_j – j-й выход; m – число выходов в устройстве,

[*] Дж. Буль - английский математик прошлого столетия, основоположник этой дисциплины.

можно описывать алгоритм работы любого устройства ЭВМ. Каждая такая зависимость y, является «булевой функцией, у которой число возможных состояний и каждой ее независимой переменной равно двум» (стандарт ISO 2382/2-76), т.е. функцией алгебры логики, а ее аргументы определены на множестве {0,1}. Алгебра логика устанавливает основные законы формирования и преобразования логических функций. Она позволяет представить любую сложную функцию в виде композиции простейших функций. Рассмотрим наиболее употребительные из них.

Известно, что количество всевозможных функций N от n аргументов выражается зависимостью

$$N=2^{2n}. \qquad (2.3)$$

При n=0 можно определить две основные функции (N=2), не зависящие от каких-либо переменных: y_0 , тождественно равную нулю ($y_0=0$), и y_1 , тождественно равную единице ($y_1=1$).

Технической интерпретацией функции $y_1=1$ может быть генератор импульсов. При отсутствии входных сигналов на выходе этого устройства всегда имеются импульсы (единицы). Функция $y_0=0$ может быть интерпретирована отключенной схемой, сигналы от которой не поступают ни к каким устройствам.

При n=1 зависимость (2.3) дает N=4. Представим зависимость значений этих функций от значения аргумента x в виде специальной таблицы истинности (таблице 2.4).

Таблица 2.4 – Таблица функций от одной переменной

Y_j x	Y_0	Y_1	Y_2	Y_3
0	0	1	0	1
1	0	1	1	0

Таблицы истинности получили такое название, потому что они определяют значение функции в зависимости от комбинации входных сигналов. В этой таблице, как и ранее, $y_0=0$ и $y_1=1$. Функция $y_2=x$, а функция $y_3=x$ – (***инверсия x***).

> Этим функциям соответствуют определенные технические аналоги. Схема, реализующая зависимость $y_2=x$, называется ***повторителем***, а схема $y_3=x$ – ***инвертором***.

При n=2, N=16, т.е. от двух переменных, можно построить шестнадцать различных функций. В таблице 2.5 представлена часть из них, имеющая фундаментальное значение при построении основных схем ЭВМ.

Таблица 2.5 – Таблица функций от двух переменных

Y_i / X1 X2	Y_0	Y_1	Y_2	Y_3	...	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9	...	Y_{15}
00	0	1	0	1		0	1	0	1	1	0		
01	0	1	0	1	...	1	0	0	1	0	1	...	
10	0	1	1	0		1	0	0	1	0	1		
11	0	1	1	0		1	0	1	0	1	0		

Заметим, что в левой части таблицы перечислены всевозможные комбинации входных переменных (наборы значений), а в правой - возможные реакции выходных сигналов. В таблице 2.5 представлены функции y_4-y_9, полностью соответствующие функциям таблице 2.4, а также новые, часто используемые и интересные функции y_4-y_9. При этом местоположение функций и их нумерация в таблице особого значения не имеют. По данной таблице нетрудно составить аналитическое выражение (зависимость) для каждой функции от двух аргументов вида (2.2). Для этого наборы переменных, на которых функция принимает значение единицы, записываются как ***конъюнкции*** (логическое умножение) и связываются знаками логического сложения. Такие формы функций получили название ***дизъюнктивных нормальных форм*** (ДНФ). Если в этих функциях конъюнкции содержат все без исключения переменные в прямом или инверсном значениях, то такая форма функций называется совершенной.

Функция y_4 представляет собой функцию логического сложения, ***дизъюнкцию***. Она принимает значение единицы, если значение единицы имеет хотя бы одна переменная x_1 или x_2:

$$y_4 = x_1\bar{x}_2 \vee \bar{x}_1 x_2 \vee x_1 x_2 = x_1 \vee x_2$$

Тождественность приведенных аналитических зависимостей можно установить, пользуясь законами алгебры логики, приведенными ниже.

Функция y_5 является ***инверсной функцией*** по отношению к y_4, она имеет название ***«отрицание дизъюнкции»***: [♦]

$$y_5 = \bar{y}_4 = \bar{x}_1 \vee \bar{x}_2 = \bar{x}_1 \cdot \bar{x}_2.$$

Функция y_6 является функцией ***логического умножения (конъюнкция)***. Она очень похожа на операцию обычного умножения и принимает значение единицы в тех случаях, когда все ее переменные равны единице:

$$y_6 = x_1 \cdot x_2.$$

[♦] Иногда в литературе встречается ее специальное название «стрелка Пирса», по фамилии математика, исследовавшего ее свойства.

Функция y_7 является инверсной функцией по отношению к y_6, она называется «***отрицание конъюнкции***» или «***штрих Шеффера***»:

$$y_7 = \overline{y}_6 = \overline{x}_1 \cdot \overline{x}_2 = \overline{x}_1 \vee \overline{x}_2$$

Функция y_8 называется ***логической равнозначностью***, она принимает значение единицы, если все ее переменные имеют одинаковое значение (или 0 или 1):

$$y_8 = \overline{x}_1 \cdot \overline{x}_2 \vee x_1 \cdot x_2 .$$

Функция y_9 является инверсной по отношению к y_8, она принимает значение единицы, если ее переменные имеют противоположные значения:

$$y_9 = \overline{y_8 = x_1 \cdot x_2 \vee x_1 \cdot x_2} = \overline{x}_1 \cdot x_2 \vee x_1 \cdot \overline{x}_2 .$$

> Ниже будет показано, что функции y_8 и y_9 являются основой для построения **сумматоров**, так как они соответствуют правилам формирования цифр двоичных чисел при сложении (вычитании).

Из перечисленных функций двух переменных можно строить сколь угодно сложные зависимости, отражающие алгоритмы преобразования информации, представленной в двоичной системе счисления. Алгебра логики устанавливает правила формирования логически полного базиса простейших функций, из которых могут строиться любые более сложные. Наиболее привычным базисом является набор трех функций {инверсия - ¬ , дизъюнкция - \/, конъюнкция - /\ или &}. Работа с функциями, представленными в этом базисе, очень похожа на использование операций обычной алгебры.

Алгебра логики устанавливает, что существуют и другие комбинации простейших логических функций, обладающих свойством логической полноты. Например, наборы логических функций {инверсия, дизъюнкция} и {инверсия, конъюнкция} также являются логически полными. Наиболее интересны минимальные базисы, включающие по одной операции {«отрицание дизъюнкции (¬ \/)»} и {«отрицание конъюнкции (¬ /\)»}. Однако работа с функциями, представленными в указанных базисах, требует от специалистов по проектированию ЭВМ определенных навыков.

2.4.2. Законы алгебры логики

Из определения вышеприведенных функций можно установить целый ряд простейших свойств:

$x \vee 1 = 1 \quad x \cdot 0 = 0$

$x \vee \overline{x} = 1 \quad x \cdot 1 = x \quad x \vee x \vee \ldots \vee x = x$

$x \vee 0 = x \quad x \cdot \overline{x} = 0 \quad x \cdot x \cdot \ldots \cdot = x$

$x \vee x = x \quad x \cdot x = x$

В алгебру логики установлен целый ряд законов, с помощью которых возможно преобразование логических функций (ЛФ).

Название закона	**Логическое выражение**
Идемпотентности	x/\x=x x\/x=x
Операции с константами	x\/1=1 x/\1=x x\/0=x x\/1=1
Операции с переменной и ее инверсией	x \/ ¬x=1 x /\ ¬x=0
коммутативный (переместительный)	x1/\x2=x2/\x1 $x_1 \vee x_2 = x_2 \vee x_1;$
ассоциативный (сочетательный)	(x1/\x2)/\x3=(x1/\x3)/\x2=x1/\(x2/\x3) $(x_1 \vee x_2) \vee x_3 = x_1 \vee (x_2 \vee x_3).$
дистрибутивный (распределительный)	$x_1 \cdot (x_2 \vee x_3) = x_1 \cdot x_2 \vee x_1 \cdot x_3$ $x_1 \vee x_2 \cdot x_3 = (x_1 \vee x_2)(x_1 \vee x_3).$
Закон поглощения	$x_1 \vee x_1 \cdot x_2 = x_1$ $x_1 \cdot (x_1 \vee x_2) = x_1.$
Закон склеивания	$x_1x_2 \vee x_1\overline{x}_2 = x_1 \quad (x_1 \vee x_2)(x_1 \vee \overline{x}_2) = x_1,$ $Fx \vee F\overline{x} = F \quad (x \vee F)(\overline{x} \vee F) = F,$
Закон свёртки	$x \vee \overline{x}F = x \vee F \quad x(\overline{x} \vee F) = xF$ где F - логическая функция общего вида, не зависящая от переменной x
Правило де Моргана	$\overline{x_1 \vee x_2} = \overline{x}_1 \cdot \overline{x}_2 \quad \overline{x_1 \cdot x_2} = \overline{x}_1 \vee \overline{x}_2.$
Закон двойного отрицания	¬ (¬x)=x

Убедиться в тождественности приведенных зависимостей можно путем аналитических преобразований выражений или путем построения таблицы истинности для ЛФ, находящихся в левой и правой частях.

Используя данные зависимости, можно преобразовывать исходные выражения в более простые (минимизировать их). По упрощенным выражениям можно построить техническое устройство, имеющее минимальные аппаратурные затраты.

2.4.3. Понятие о минимизации логических функций

Проблема минимизации логических функций решается на основе применения законов склеивания и поглощения с последующим перебором получаемых дизъюнктивных форм и выбором из них оптимальной (минимальной). Существует большое количество методов минимизации ЛФ. Все они отличаются друг от друга спецификой применения операций склеивания и поглощения, а также различными способами сокращения переборов. Среди аналитических методов наиболее известным является метод Квайна-Маккласки, среди табличных - метод с применением диаграмм Вейча и карт Карно. Графические методы минимизации отличаются большей наглядностью и меньшей трудоемкостью. Однако их применение эффективно при малом числе переменных n<5.

Рассмотрим последовательность действий минимизации ЛФ на примере.

Пример 2.15. Найти минимальную дизъюнктивную форму функции, заданной таблицей истинности (таблица 2.6).

Таблица 2.6 – Таблица истинности функции Y = f (X_1,X_2,X_3)

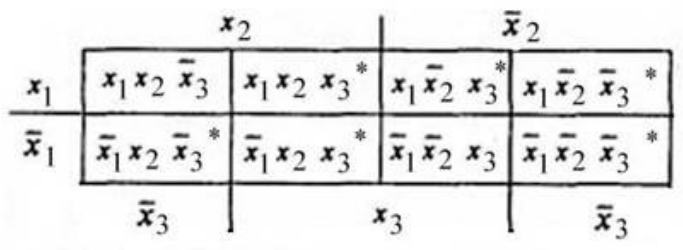
X_1	X_2	X_3	Y	Логическое выражение
0	0	0	1	$\bar{x}_1 \wedge \bar{x}_2 \wedge \bar{x}_3$
0	0	1	0	$\bar{x}_1 \wedge \bar{x}_2 \wedge x_3$
0	1	0	1	$\bar{x}_1 \wedge x_2 \wedge \bar{x}_3$
0	1	1	1	$\bar{x}_1 \wedge x_2 \wedge x_3$
1	0	0	1	$x_1 \wedge \bar{x}_2 \wedge \bar{x}_3$
1	0	1	1	$x_1 \wedge \bar{x}_2 \wedge x_3$
1	1	0	0	$x_1 \wedge x_2 \wedge \bar{x}_3$
1	1	1	1	$x_1 \wedge x_2 \wedge x_3$

Эта функция интересна тем, что имеет несколько минимальных форм. По данным таблицы запишем аналитическое выражение:

$$y = \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1 x_2 \bar{x}_3 \vee \bar{x}_1 x_2 x_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1\bar{x}_2 x_3 \vee x_1 x_2 x_3$$

Штриховыми линиями в этом выражении отмечены пары конъюнкций, к которым можно применить операцию склеивания типа $F_x \wedge F_{\bar{x}}$. Особенно это видно при использовании диаграммы Вейча, в которой «склеиваемые» конъюнкции находятся по соседству друг с другом. Диаграмма Вейча просто по-другому интерпретирует таблицу истинности (таблица 2.7).

Таблица 2.7 - Диаграмма Вейча функции Y



После выделения конъюнкций (они отмечены звездочкой), видно, какие конъюнкции могут образовывать пары для склеивания.

В результате применения операций склеивания и поглощения можно получить другое аналитическое выражение:

$$y = \bar{x}_1 x_2 \vee x_2 x_3 \vee x_1 x_3 \vee x_1 \bar{x}_2 \vee \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_3,$$

в котором отсутствуют возможности дальнейших склеивании и поглощений. Однако последнее выражение является избыточным, так как отдельные конъюнкции могут быть «лишними», т.е. их «составные части» могут включаться в другие конъюнкции. У данной функции существует пять безызбыточных дизъюнктивных форм, из которых только две являются минимальными:

$$y_1 = \bar{x}_1 x_2 \vee x_1 x_3 \vee \bar{x}_2 \bar{x}_3;$$
$$y_2 = \bar{x}_1 x_2 \vee x_2 x_3 \vee x_1 \bar{x}_2 \vee \bar{x}_1 \bar{x}_3;$$
$$y_3 = \bar{x}_1 x_2 \vee x_1 x_3 \vee x_1 \bar{x}_2 \vee \bar{x}_1 \bar{x}_3;$$
$$y_4 = \bar{x}_1 \bar{x}_3 \vee x_2 x_3 \vee x_1 \bar{x}_2;$$
$$y_5 = \bar{x}_1 \bar{x}_3 \vee \bar{x}_1 x_2 \vee x_1 x_3 \vee x_1 \bar{x}_2$$

Из приведенных зависимостей видно, что только функции y_1 и y_4 являются минимальными формами функций, так как они содержат наименьшее число конъюнкций и имеют минимальный ранг этих конъюнкций.

Данный пример показывает, что диаграммы Вейча не дают сразу самый оптимальный вариант для построения ДНФ. Лучше использовать другой табличный метод – метод Карно (карты Карно).

$x_1 x_2$ / x_3	00	01	10	11
0	1	1	0	1
1	0	1	1	1

Рассмотрим алгоритм построения ДНФ. Внутри таблицы (карты) выбираются блоки, содержащие максимальное количество подряд идущих 1 (их количество должно быть степенью 2), таким образом, чтобы покрыть все 1, присутствующее в таблице (при построении блоков допускается их пересечение, но не поглощение).

Замечание: Чем больше размер блока, тем короче логическое выражение.

Внутри блока выражение строится через конъюнкцию, блоки между собой объединяются через дизъюнкцию.

Таким образом, для приведенного примера ДНФ будет иметь вид:

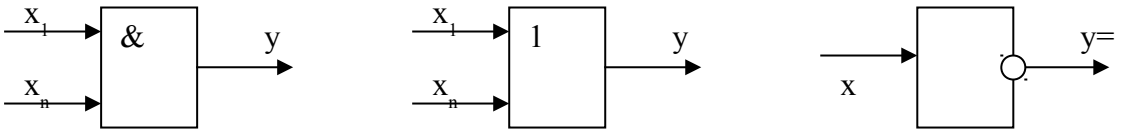
$y = (\overline{x_3} \wedge \overline{x_1}) \vee (x_3 \wedge x_2) \vee (x_1 \wedge \overline{x_2})$

$y = (\overline{x_3} \wedge \overline{x_2}) \vee (x_3 \wedge x_1) \vee (\overline{x_1} \wedge x_2)$, что совпадает с предыдущим решением.

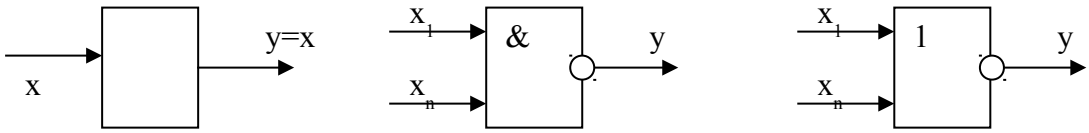
Минимизация «вручную» возможна только для функций, зависящих от 4-5 переменных, так как трудоемкость переборов растет в квадратичной зависимости от числа переменных. Применение мощных ЭВМ для этих Целей позволяет расширить границы до n= 12-15. Если при этом учесть, что функции могут быть частично определены (значения функций на некоторых наборах переменных можно определять произвольно), а также что иногда приходится решать задачи совместной минимизации систем ЛФ, то минимизация ЛФ становится сложной инженерной, практической и научной проблемой.

2.4.4. Техническая интерпретация логических функций

Каждой логической операции соответствует свой аппаратный блок (см. п.2.4.1.). На рисунке 2.2. приведены базовые блоки: повторитель, конъюнктор (И-&), дизъюнктор (ИЛИ-1), инвертор (НЕ - ¬), отрицание конъюнкции (И-НЕ), отрицание дизъюнкции (ИЛИ-НЕ).



а) Конъюнктор б) Дизъюнктор в) Инвертор



г) Повторитель д) И-НЕ е) ИЛИ-НЕ

Рисунок 2.2 - Обозначение логических элементов на схеме

По логическим выражениям проектируются схемы ЭВМ. При этом следует придерживаться следующей последовательности действий.

1. Записать словесное описание работы схемы.

2. Формализовать словесное описание, записав функции в дизъюнктивной (конъюнктивной) совершенной нормальной форме по таблицам истинности.
3. Минимизировать логические зависимости с целью их упрощения.
4. Представить полученные выражения в выбранном логически полном базисе элементарных функций.
5. Построить схему устройства.
6. Проверить работоспособность полученной схемы.

ГЛАВА 3 КЛАССИФИКАЦИЯ ЭЛЕМЕНТОВ И УЗЛОВ ЭВМ

3.1. Классификация элементов и узлов ЭВМ

При рассмотрении структуры любой ЭВМ обычно проводят ее детализацию. Как правило, в структуре ЭВМ выделяют следующие структурные единицы:

1. устройства,
2. блоки,
3. узлы,
4. элементы.

Такая детализация соответствует вполне определенным операциям преобразования информации, заложенным в программах пользователей.

Нижний уровень обработки реализуют ***элементы***. Каждый элемент предназначается для обработки единичных электрических сигналов, соответствующих битам информации. ***Узлы*** обеспечивают одновременную обработку группы сигналов - информационных слов. ***Блоки*** реализуют некоторую последовательность в обработке информационных слов - функционально обособленную часть машинных операций (блок выборки команд, блок записи-чтения и др.). ***Устройства*** предназначаются для выполнения отдельных машинных операций и их последовательностей.

В общем случае любая структурная единица ЭВМ обеспечивает преобразование входной информации X в выходную У (см. рисунок 2.1). Все современные вычислительные машины строятся на комплексах системах интегральных микросхем (ИС).

> ***Электронная микросхема называется интегральной***, если ее компоненты и соединения между ними выполнены в едином технологическом цикле, на едином основании и имеют общую герметизацию и защиту от механических воздействий.

Каждая микросхема представляет собой миниатюрную электронную схему, сформированную послойно в кристалле полупроводника: кремния, германия и т.д. В состав микропроцессорных наборов включаются различные типы микросхем, но все они должны иметь единый тип межмодульных связей, основанный на стандартизации параметров сигналов взаимодействия (амплитуда, полярность, длительность импульсов и т.п.). Основу набора обычно составляют большие БИС и даже сверхбольшие интегральные схемы. На очереди следует ожидать появления ультра больших ИС (УБИС). Кроме них обычно используются микросхемы с малой и средней степенью интеграции

(СИС). Функционально микросхемы могут соответствовать устройству, узлу или блоку, но каждая из них состоит из комбинации простейших логических элементов, реализующих функции формирования, преобразования, запоминания сигналов и т.д.

Элементы ЭВМ можно классифицировать по различным признакам. Наиболее часто такими *признаками* являются:

- тип сигналов,
- назначение элементов,
- технология их изготовления и т.д.

В ЭВМ широко применяют два способа физического представления сигналов: *импульсный* и *потенциальный*. При импульсном способе представления сигналов единичному значению некоторой двоичной переменной ставится в соответствие наличие импульса (тока или напряжения), нулевому значению - отсутствие импульса (рисунок 3.1, *а).* Длительность импульсного сигнала не превышает один такт синхроимпульсов.

При потенциальном или статическом представлении сигналов единично значение двоичной переменной отображается высоким уровнем напряжения, а нулевое значение - низким уровнем (рисунок 3.1, б).

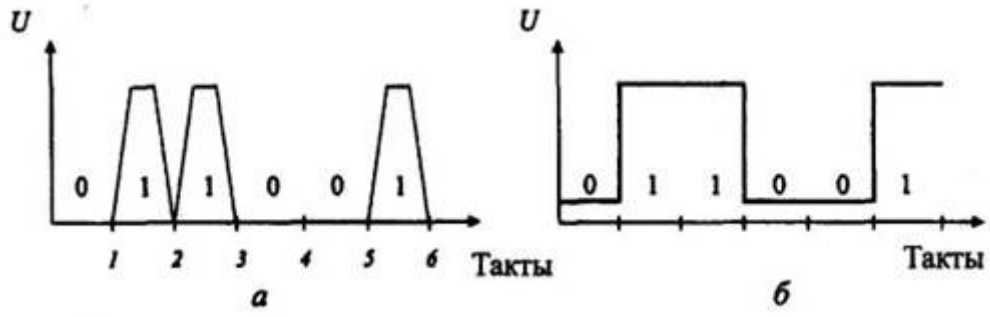


Рисунок 3.1 - Представление информации в ЭВМ:
а - импульсные сигналы; *б* - потенциальные сигналы

Независимо от вида сигналов различают последовательный и параллельный коды передачи и представления информации в ЭВМ.

При последовательном коде представления данных используются одиночные шины или линии передачи, в которых сигналы, соответствующие отдельным разрядам данных, разнесены во времени. Обработка такой информации производится последовательно разряд за разрядом. Такой вид представления и передачи данных требует весьма экономичных по аппаратурным затратам схем обработки данных. Время же обработки определяется числом обрабатываемых сигналов (разрядов).

Параллельный код отображения и передачи информации предполагает параллельную и одновременную фиксацию всех разрядов данных на различных шинах,

т.е. параллельный код данных развернут в пространстве. Это дает возможность ускорить обработку во времени, но затраты на аппаратурные средства при этом возрастают пропорционально числу обрабатываемых разрядов.

Во всех вычислительных машинах используются и ***параллельно-последовательные коды*** представления информации. При этом информация отображается частями. Части поступают на обработку последовательно, а каждая часть данных представляется параллельным кодом.

По своему назначению элементы делятся на *формирующие*, *логические* и *запоминающие*.

К ***формирующим элементам*** относятся различные формирователи, усилители, усилители-формирователи и т.п. Данные элементы служат для выработки определенных электрических сигналов, восстановления их параметров (амплитуды, полярности, мощности, длительности).

В каждой ЭВМ имеются специальные блоки, формирующие сигналы тактовой частоты, серии синхронизирующих и управляющих сигналов, координирующих работу всех схем ЭВМ. Длительность такта является важной характеристикой ЭВМ, определяющей ее потенциальную производительность. Время выполнения любой операции ЭВМ связано с определенным числом тактов.

Интервал времени между импульсами основной частоты называется ***тактом***.

Простейшие ***логические элементы*** преобразуют входные сигналы в соответствии с элементарными логическими функциями, рассмотренными в п.2.4. В свою очередь, полученные сигналы могут формировать следующий уровень сигналов и т. д. Сложные преобразования в соответствии с требуемыми логическими зависимостями могут приводить к построению многоуровневых схем. Каждая такая схема представляет собой композицию простейших ***логических схем***.

Запоминающим элементом называется элемент, который способен принимать и хранить код двоичной цифры (единицы или нуля).

Элементы памяти могут запоминать и сохранять исходные значения некоторых величин, промежуточные значения обработки и окончательные результаты вычислений. Только *запоминающие элементы* в схемах ЭВМ позволяют проводить обработку информации с учетом ее развития.

3.2. Комбинационные схемы

Обработка входной информации X в выходную У (см. рисунок 2.1) в любых схемах ЭВМ обеспечивается преобразователями или цифровыми автоматами двух видов: ***комбинационными схемами*** и ***схемами с памятью***.

> ***Комбинационные схемы*** (КС) – это схемы, у которых выходные сигналы $Y = (y_1, y_2, ..., y_m)$ в любой момент дискретного времени однозначно определяются совокупностью входных сигналов $X = (x_1, x_2,..., x_n)$, поступающих в тот же момент времени *t*.

Реализуемый в КС способ обработки информации называется комбинационным потому, что результат обработки зависит только от комбинации входных сигналов и формируется сразу при поступлении входных сигналов. Поэтому одним из достоинств комбинационных схем является их высокое быстродействие. Преобразование информации однозначно описывается логическими функциями вида ***Y=f(X)***.

Логические функции и соответствующие им комбинационные схемы подразделяют на ***регулярные*** и ***нерегулярные структуры***. Регулярные структуры предполагают построение схемы таким образом, что каждый из ее выходов строится по аналогии с предыдущими. В нерегулярных структурах такая аналогия отсутствует.

В практике проектирования ЭВМ накоплен огромный опыт по синтезу различных схем. Многие регулярные структуры положены в основу построения отдельных ИС малой и средней степени интеграции или отдельных функциональных частей БИС и СБИС. Из *регулярных комбинационных схем* наиболее распространены:

- дешифраторы,
- шифраторы,
- схемы сравнения,
- комбинационные сумматоры,
- коммутаторы и др.

Рассмотрим принципы построения подобных регулярных структур.

Дешифраторы

Дешифраторы (ДШ) - это комбинационные схемы с n входами и $m = 2^n$ выходами и преобразующая двоичный код на своих входах в *унитарный код* на выходах.

> ***Унитарным*** называется ***двоичный код***, содержащий одну и только одну единицу, например 00100000.

Единичный сигнал, формирующийся на одном из *m* выходов, однозначно соответствует комбинации входных сигналов. Например, разработка структуры ДШ для *n=3* согласно методике, изложенной в п.2.4, позволяет получить таблицу истинности (таблица 3.1) и логические зависимости.

Таблица 3.1 - Таблица истинности дешифратора

Входы			Выходы					
X_2	X_1	X_0	Y_0	Y_1	...	Y_5	Y_6	Y_7
0	0	0	1	0		0	0	0
0	0	1	0	1		0	0	0
0	1	0	0	0		0	0	0
0	1	1	0	0	…	0	0	0
1	0	0	0	0		0	0	0
1	0	1	0	0		1	0	0
1	1	0	0	0		0	1	0
1	1	1	0	0		0	0	1

Условно-графическое обозначение *дешифратора* на три входа приведено на рисунке 3.1.

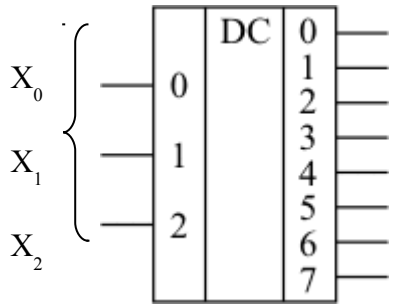


Рисунок 3.1 – **У**словно-графическое обозначение трехвходового дешифратора

Рассмотрим логические зависимости при формировании сигнала на одном из его выходов (например, сигнал f_5 на выходе 5): $\overline{f_5} = \overline{x_0} \vee x_1 \vee \overline{x_2}$.

X_1X_2 / X_0	00	01	10	11
0	0	0	0	0
1	0	1	0	0

Реализация ДШ в одноэлементном базисе «Штрих Шеффра» $f_5 = \overline{\overline{x_0} \vee x_1 \vee \overline{x_2}} = x_0 \wedge \overline{x_1} \wedge x_2$ достаточно проста (см. рисунок 3.2).

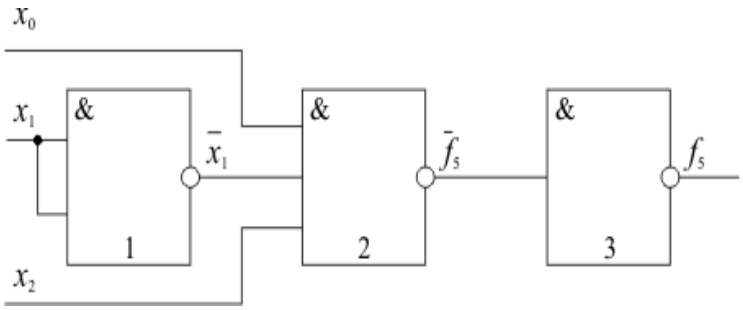


Рисунок 3.2 – Схема формирования сигнала на 5 выходе ДШ

Из представленной схемы видно, что фактически логику преобразования выполняет лишь элемент 2, в то время как элемент 1 служит для получения инверсии сигнала x_1, а элемент 3 преобразует полученное на элементе 2 инверсное значение функции в прямое. Многие элементы хранения, например триггерные схемы, позволяют получать сигнал в ***парафазном коде***, то есть имеют два выхода, на одном из которых сигнал имеет прямое, а на другом – инверсное значение. Это позволяет избавиться от элемента 1 в схеме. Если предположить, что значения выходных сигналов имеют инверсный вид по отношению к представленному в таблице 3.1, то отпадает необходимость в элементе 3. В большинстве реальных интегральных микросхем реализованы именно *дешифраторы* с инверсными выходами. Обозначение такого *дешифратора* показано на рисунке 3.3.

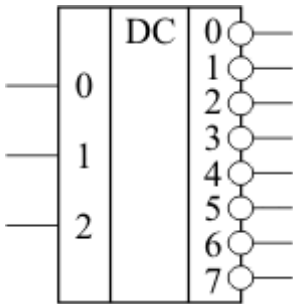


Рисунок 3.3 – Условно-графическое обозначение дешифратора с инверсными выходами

На выходах такого *дешифратора* образуется унитарный код, содержащий один и только один ноль. Например, если входные сигналы имеют значение $110_2=6_{10}$, то выходы *дешифратора*, представленного на рисунке 3.3, будут находиться в состоянии 10111111, то есть выход 6 будет иметь значение, отличное от остальных выходов.

Дешифраторы широко применяются в различных устройствах компьютеров. Прежде всего, они используются для выбора ячейки запоминающего устройства, к которой производится обращение для записи или считывания информации, для расшифровки кода операции и т.д. При этом часть разрядов адресного кода может дешифрироваться *дешифраторами*, выполненными в виде отдельных интегральных схем,

а другая часть разрядов (обычно младшая) дешифрируется с помощью *дешифраторов*, встроенных непосредственно в БИС запоминающего устройства. Кроме того, *дешифраторы* находят применение в устройстве управления для определения выполняемой операции, построения распределителей импульсов и в других блоках.

Шифратор

Шифратор – схема, имеющая 2^n входов и n выходов, функции которой во многом противоположны функции дешифратора (рисунок 3.4). Эта комбинационная схема в соответствии с унитарным кодом на своих входах формирует *позиционный код* на выходе (таблица 3.2).

Таблица 3.2 – Таблица истинности шифратора

Входы				Выходы	
X_3	X_2	X_1	X_0	y_0	y_1
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	1	1	1

Обратите внимание, что таблицы 3.1 и 3.2 очень похожи, только входы и выходы поменялись местами.

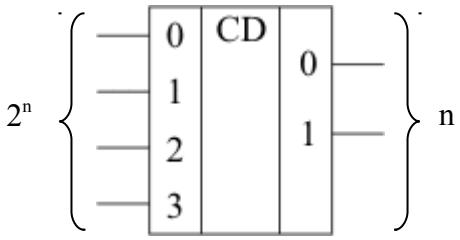


Рисунок 3.4 – Условно-графическое обозначение шифратора на 4 входа

Компаратор

Компараторы – это комбинационные схемы, осуществляющие сравнение двух чисел. Результатом сравнения является обнаружение одного из трех состояний: A=B, A>B, A<B.

Компаратор *(схемы сравнения)* обычно строятся как поразрядные. Они широко используются и автономно, и в составе более сложных схем, например, при построении сумматоров.

Таблица истинности (таблица 3.3) отражает логику работы 1-го разряда схемы сравнения при сравнении двух векторов A и B. На рисунке 3.5 показана структурная схема компаратора.

Таблица 3.3 – Таблица истинности компаратора

Входы		Выходы		
A_i	B_i	$Y_{A=B}$	$Y_{A>B}$	$Y_{A<B}$
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

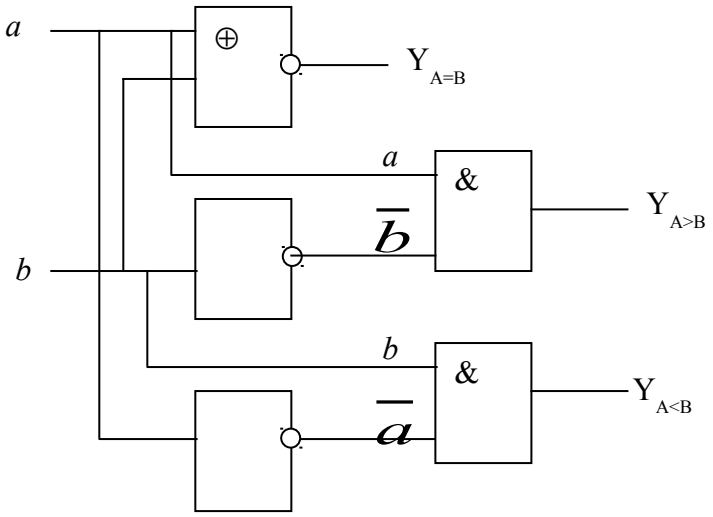


Рисунок 3.5 – Схема работы компаратора

3.2. Схемы с памятью

Более сложным преобразователем информации являются *схемы с памятью*. Наличие памяти в схеме позволяет запоминать промежуточные состояния обработки и учитывать их значения в дальнейших преобразованиях. Выходные сигналы $Y = (y_1, y_2, ..., y_m)$ в схемах данного типа формируются не только по совокупности входных сигналов $X = (x_1, x_2, ..., x_n)$, но и по совокупности состояний схем памяти $Q = (q_1, q_2, ..., q_k)$. При этом различают текущий дискретный момент времени t и последующий (t+1) момент времени (рисунок 3.6).

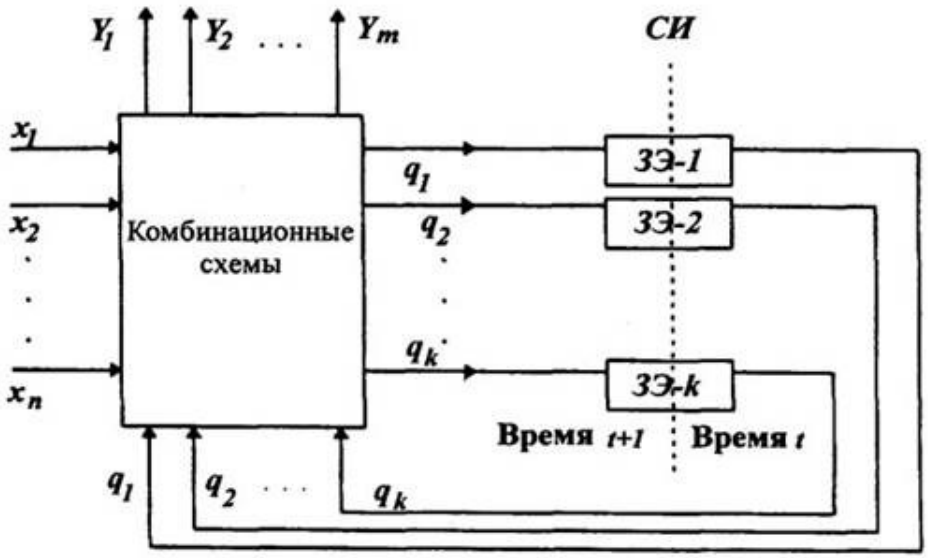


Рисунок 3.6 – Обобщенная структура схемы с памятью

Передача значения Q между моментами времени t и $(t+1)$ осуществляется обычно с применением двухступенчатой памяти и синхронизирующих импульсов (СИ).

В качестве простейшего запоминающего элемента (ЗЭ) в современных ЭВМ используют *триггеры*. В связи с успешным применением микроэлектроники в схемах основных устройств ЭВМ (процессоров и оперативной памяти) исчезли в качестве запоминающихся элементов схемы, использующие остаточную намагниченность - ферритовые сердечники. Самая простейшая схема триггера может быть синтезирована по общим правилам (п.2.4.4).

Триггер

Триггер – электронная схема, обладающая двумя устойчивыми состояниями 0 и 1. Переход из одного устойчивого состояния в другое происходит скачкообразно под воздействием управляющих сигналов. При этом также скачкообразно изменяется уровень напряжения на выходе триггера. *Триггеры* служат основой для построения *регистров*, *счетчиков* и других элементов, обладающих *функцией хранения*.

Триггеры можно классифицировать по различным признакам, например, так, как это показано на рисунке 3.7. Структура триггера определяется в зависимости от элементной базы и таблицы переходов, которая является модификацией таблицы истинности.

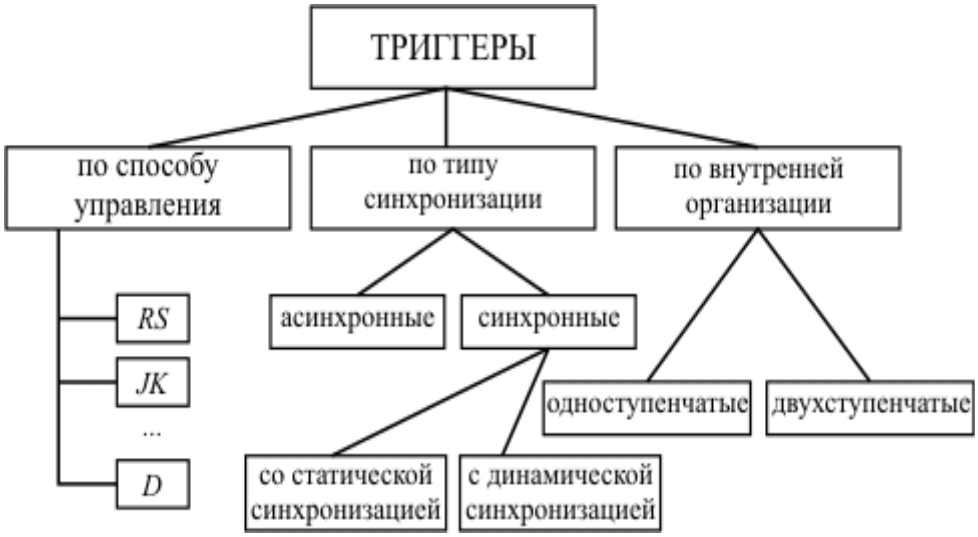


Рисунок 3.7 – Классификация триггерных схем

Главной частью любого *триггера* является запоминающая ячейка (ЗЯ). Схема запоминающей ячейки на элементах "И-НЕ" представлена на рисунке 3.8.

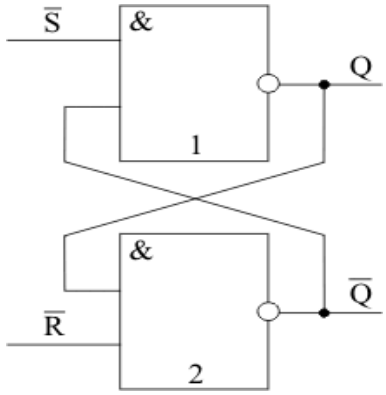


Рисунок 3.8 – Запоминающая ячейка на элементах «И-НЕ»

Входной сигнал **S** (Set) служит для установки ЗЯ в состояние «1» (Q=1, $\overline{Q}=0$). Сигнал **R** (Reset) устанавливает ЗЯ в состояние «0» (Q=0, $\overline{Q}=1$). Активными значениями для них являются сигналы низкого уровня.

RS-триггеры

Рассмотрим таблицу переходов для *асинхронного RS-триггера* (таблица 3.4).

Таблица 3.4 - Условия работы асинхронного триггера

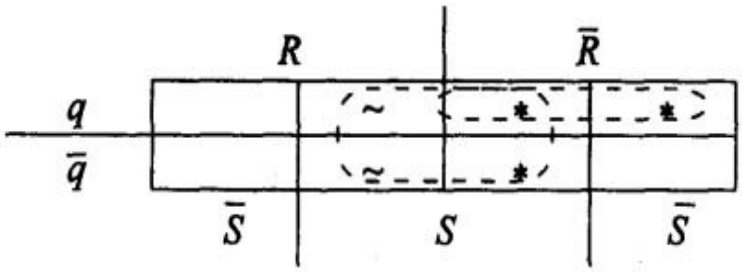
Входы		**Состояние q_{t+1}**		
R	**S**	**0**	**1**	Режим
0	0	0	1	Хранение
1	0	0	0	Установка 0
0	1	1	1	Установка 1
1	1	?	?	Запрещенное состояние

Содержание таблицы расшифровывается следующим образом. Элемент памяти может сохранять значение q_t=0 или q_t=1 в зависимости от установки ранее установленного

состояния. При отсутствии входных сигналов на входах R и S (R =0 и S =0) значения q_{t+1} первой строке таблицы в точности повторяют значения q_t. При поступлении сигнала R=l (сигнала установки «нуля») элемент независимо от своего состояния принимает значение, равное нулю, q_{t+1}=0. Если же на вход S поступает сигнал установки «единицы» (S=1), то q_{t+1}=1 независимо от предыдущего состояния q_t. Одновременное поступление сигналов на входы R и S является запрещенной ситуацией, так как она может привести к непредсказуемому состоянию. В схемах формирования сигналов R и S должны быть предусмотрены блокировки, исключающие их совпадения, S=R=1.

Для таблицы переходов (таблицы 3.4) может быть построена диаграмма Вейча (таблица 3.5).

Таблица 3.5 – Диаграмма Вейча для таблицы переходов триггера



В этой таблице знаком "~"отмечены запрещенные комбинации входных сигналов. Эти комбинации могут быть использованы для упрощения логических зависимостей. Логическая зависимость, описывающая работу элемента памяти, принимает вид:

$$q_{t+1} = \bar{R}_t\bar{S}_t q_t \vee \bar{R}_t S_t\bar{q}_t \vee \bar{R}_t S_t q_t = \bar{R}_t S_t \vee q_t\bar{R}_t. \qquad (3.1)$$

Уравнение (3.1) получено путем эквивалентных преобразований. Добавление в него комбинаций, соответствующих запрещенным ситуациям и помеченных знаком "~", т.е.

$$R_t S_t q_t \vee R_t S_t\bar{q}_t = R_t S_t,$$

позволяет еще больше упростить уравнение триггера:

$$q_{t+1} = \bar{R}_t S_t \vee q_t\bar{R}_t \vee R_t S_t = S_t(\bar{R}_t \vee R_t) \vee q_t\bar{R}_t = S_t \vee q_t \cdot \bar{R}_t. \qquad (3.2)$$

Для реализации полученной зависимости в базисе «И-НЕ» применим правило де Моргана и получим функцию

$$q_{t+1} = \overline{\overline{S_t \vee q_t\bar{R}_t}} = \overline{\bar{S} \cdot \overline{q_t\bar{R}_t}}.$$

По данной зависимости можно построить схему элемента памяти - асинхронного RS-тригера. В этой схеме следует только соединить выход q_{t+1} со входом q_t. На рисунке 3.9 эта связь отмечена штриховой линией.

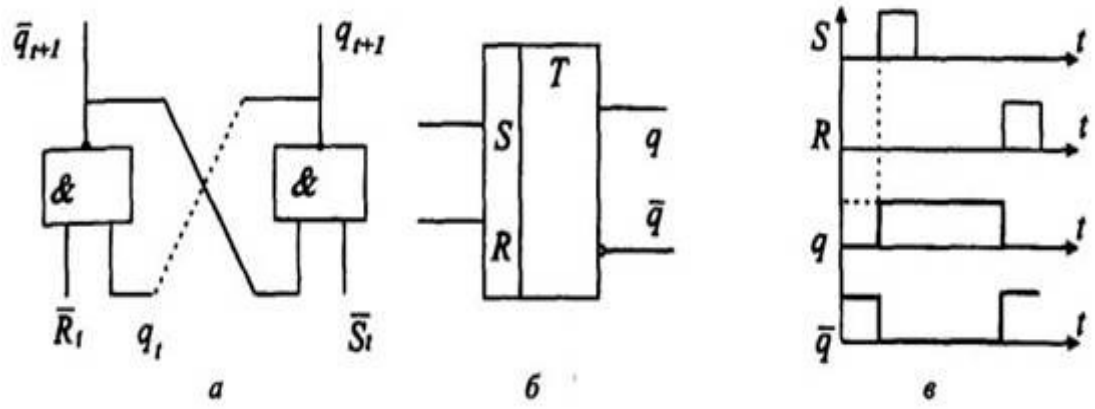


Рисунок 3.9 – Схема асинхронного RS-триггера:
а- схема; *б* - обозначение на принципиальных электрических схемах;
в - временная диаграмма

RS-триггер нашел широкое распространение в схемах ЭВМ. Одиночные триггеры этого типа часто используются в различных блоках управления. В асинхронных RS-триггерах имеется один существенный недостаток, обусловленный самой логикой их построения (см. таблицу 3.4), т.е. в них сигналы R и S должны быть разнесены во времени. Дополнение этого триггера комбинационными схемами синхронизации на входе и выходе позволяет получить триггеры с более сложной логикой работы: синхронные RS-триггеры, T-, JK-, D- триггеры и целый ряд комбинированных RST-, JKRS-, DRS-триггеров.

- Прописные буквы в названиях триггеров обозначают:
- **R** (Reset - сброс) - вход установки триггера в нулевое состояние Q=0;
- **S** (Set - установка) - вход установки триггера в единичное состояние Q=1;
- **T** (Toggle - релаксатор) - счетный вход триггера;
- **J** (Jerk - внезапное включение) - вход установки JK-триггера в единичное состояние Q=1;
- **K** (Kill - внезапное выключение) - Q=0;
- D (Delay - задержка) - вход установки триггера в единичное или нулевое состояние на время, равное одному такту;
- C (Clock - часы) - вход синхронизирующих тактовых импульсов.

Триггер называется синхронным, если его таблица переходов хотя бы по одному управляющему входу реализуется под воздействием синхронизирующего сигнала. Обобщенная схема синхронного одноступенчатого триггера приведена на рисунке 3.10.

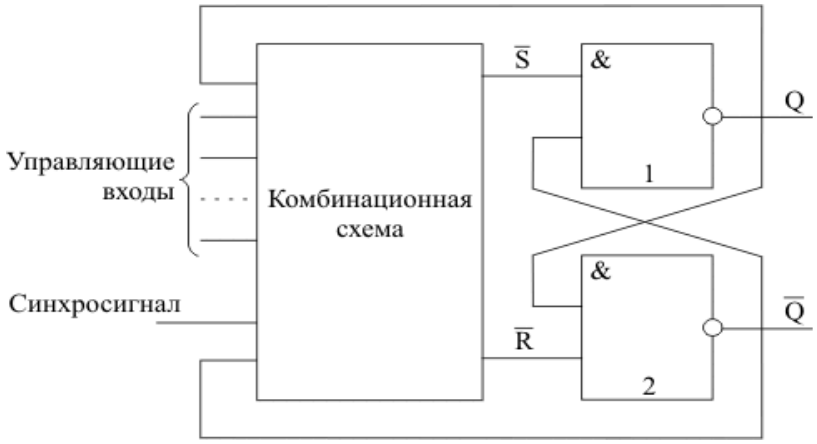


Рисунок 3.10 – Обобщенная схема синхронного одноступенчатого триггера

Основу синхронного одноступенчатого триггера составляет рассмотренная выше запоминающая ячейка (элементы 1, 2). Комбинационная схема преобразует управляющие сигналы триггера, а также, для некоторых типов триггеров, сигналы Q и Q с выходов ЗЯ в сигналы S и R на входах запоминающей ячейки. Синхросигнал C разрешает передачу на входы элементов 1 и 2 таких значений сигналов S и R, которые устанавливают ЗЯ в то или иное состояние. Неактивное значение синхросигнала обеспечивает на входах запоминающей ячейки состояние управляющих сигналов S = R = 1, что соответствует для нее режиму хранения.

На рисунке 3.11 показаны схемы синхронного однотактного *(а)* и двухтактного *(б)* RS-триггеров.

Двухкаскадная схема RS-триггера (рисунок 3.11, б) нашла наиболее широкое применение для построения *n*-разрядных схем запоминания - всевозможных *регистровых схем*. Штриховыми линиями на схеме указаны дополнительные точки подключения сигналов установки и сброса.

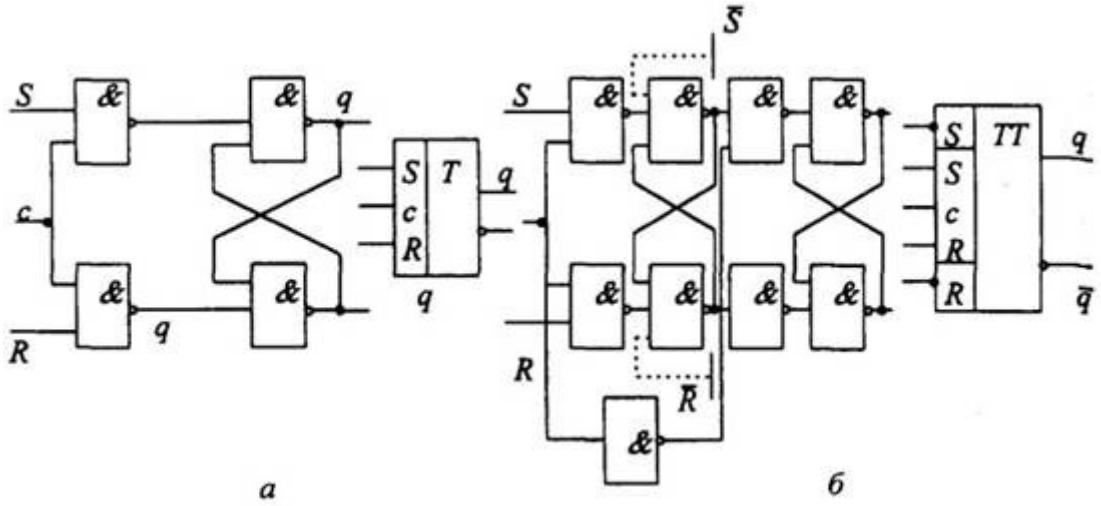


Рисунок 3.11 – Электрическая и функциональная схемы синхронных RS-тригтеров:

а, б - варианты

Представленный на рисунке 3.11 а) триггер имеет *статическую синхронизацию*, при которой управляющие сигналы активизируют входы S и R запоминающей ячейки во время высокого уровня сигнала на входе синхронизации. Его условно-графическое обозначение приведено на рисунке 3.12. Условно-графические обозначения *триггеров*, использующих другие типы синхронизации, приведены на рисунке 3.12 б), в), г) (на примере *RS-триггера*).

На рисунке 3.12 б) представлено условно-графическое обозначение *триггера* со статической синхронизацией в случае, если активный уровень синхросигнала низкий. Условно-графические обозначения *триггеров* с динамической синхронизацией показаны на рисунке 3.12 в), г). В первом случае изменение состояния *триггера* под воздействием поступивших управляющих сигналов происходит только в момент переключения синхронизирующего сигнала с низкого уровня на высокий, а во втором – при переключении с высокого на низкий уровень. При постоянном значении уровня синхросигнала состояние выхода *триггера* с динамической синхронизацией не меняется при любых изменениях управляющих сигналов на его входах.

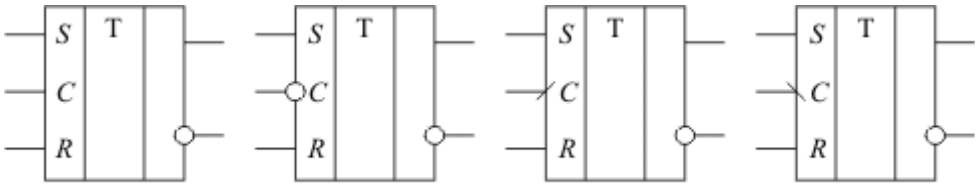


Рисунок 3.12 – Условно-графические обозначения RS-триггера с различной синхронизацией:
а - статическая синхронизация; б - статическая инверсная синхронизация;
в - динамическая синхронизация передним фронтом синхросигнала;
г - динамическая синхронизация задним фронтом синхросигнала

Идеализированная (без учета задержек) временная диаграмма работы *RS-триггеров* с различными типами синхронизации приведена на рисунке 3.13.

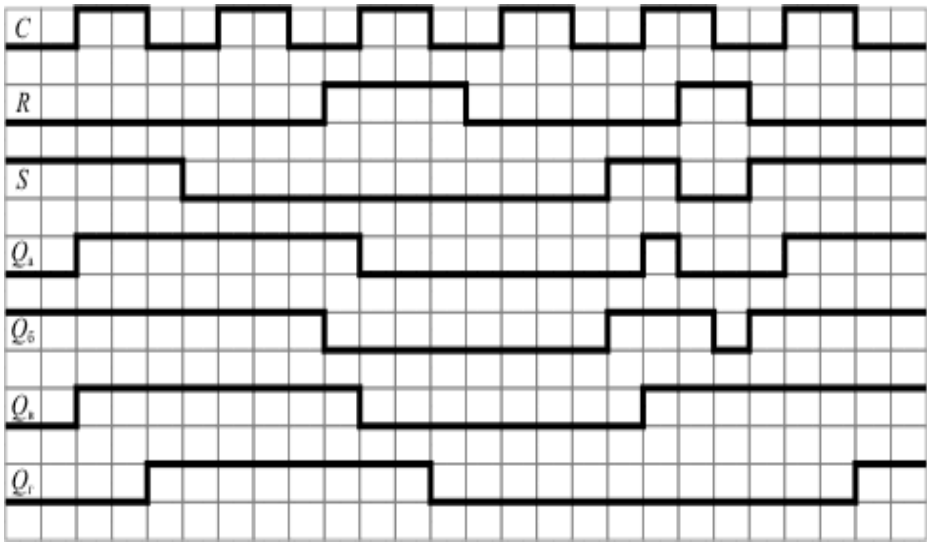


Рисунок 3.13 – Временная диаграмма работы RS-триггера с различными типами

синхронизации: Qа – статическая синхронизация; Qб– статическая инверсная синхронизация; Qв –динамическая синхронизация передним фронтом синхросигнала; Qг– динамическая синхронизация задним фронтом синхросигнала

Как отмечалось выше, синхронный триггер, помимо управляющих входов, воздействующих на его состояние при подаче сигнала синхронизации, может иметь входы, которые воздействуют на его состояние непосредственно. Обычно они используются для установки триггера в то или иное начальное состояние перед подачей последовательности синхросигналов. Схема синхронного RS-триггера с асинхронными входами установки в «0» и в «1» приведена на рисунке 3.14, а его условно-графическое обозначение – на рисунке 3.15.

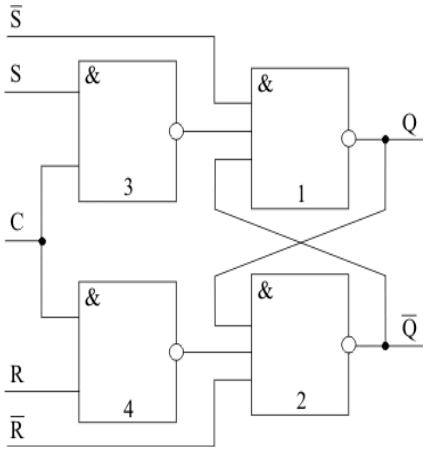


Рисунок 3.14 – Синхронный одноступенчатый RS-триггер с асинхронными установочными входами

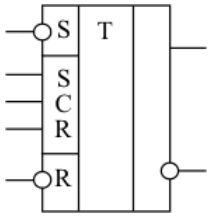


Рисунок 3.15 – Условно-графическое обозначение синхронного одноступенчатого RS-триггера с асинхронными установочными входами

Сигналы, поступающие по асинхронным входам S и R, подаются непосредственно на входы запоминающей ячейки, образованной элементами 1 и 2, минуя цепь, управляемую синхросигналом (элементы 1 и 2), и вызывают переключение запоминающей ячейки согласно таблице 3.4.

Триггеры некоторых типов используют значения выходного сигнала для формирования управляющих сигналов на входах запоминающей ячейки (см. рисунок 3.10). Это может привести к непредсказуемой последовательности его переключений. При построении отдельных схем на основе *триггеров*, например, *регистров сдвига*, необходимо, чтобы значения выходных сигналов *триггера* не

изменялись на то время, пока производится их запись и значения его выходных сигналов в другой *триггер*, но сам этот *триггер* должен воспринимать значения с выхода некоторой другой триггерной схемы. Эти, а также некоторые другие ситуации требуют особых подходов к организации *триггера*, основным из которых является создание *двухступенчатых триггеров*.

Двухступенчатый RS-триггер

Двухступенчатый RS-триггер (рисунок 3.16 и рисунок 3.17) строится на основе двух одноступенчатых *триггеров* с прямой статической синхронизацией. Информация в первую ступень *триггера* (элемент 1) заносится во время действия высокого уровня синхросигнала. После того как синхросигнал на входе принимает низкое значение, элемент 1 переходит в режим хранения, а значение высокого сигнала на выходе инвертора 3 обеспечивает запись состояния *триггера* 1 в *триггер* 2. Идеализированная временная диаграмма работы *двухступенчатого RS-триггера* приведена на рисунке 3.18.

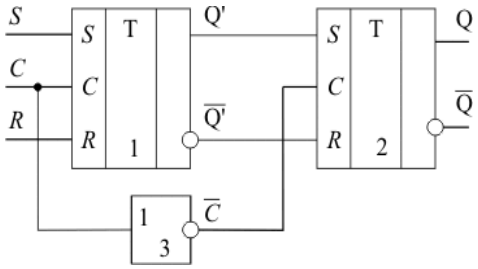


Рисунок 3.16 – Схема двухступенчатого RS-триггера

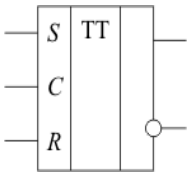


Рисунок 3.17 – Условно-графическое обозначение двухступенчатого RS-триггера

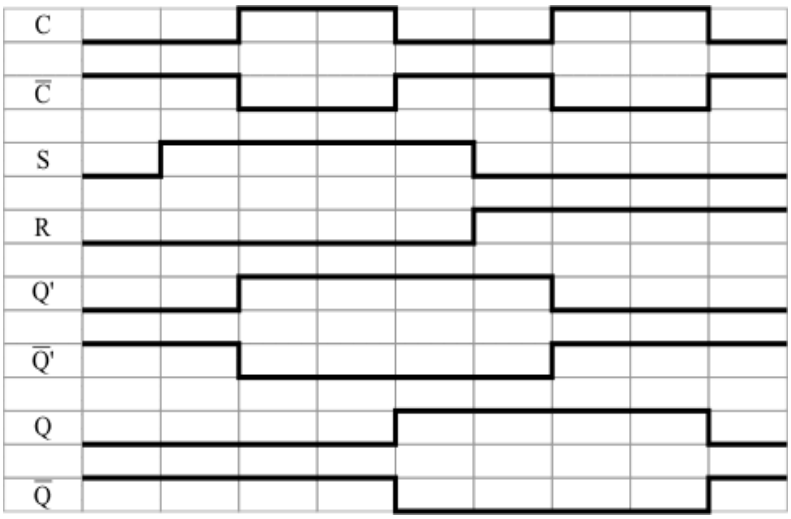


Рисунок 3.18 – Временная диаграмма работы двухступенчатого RS-триггера

Т-триггер

На рисунке 3.19 приведена схема **Т-триггера** или иначе - *триггера со счетным входом*. При значении *Т=0* триггер сохраняет свое ранее установленное состояние - режим хранения состояния, при *Т=1* триггер переходит в противоположное состояние. Таблица переходов (таблица 3.6) и диаграмма работы (рисунок 3.19, *б)* отражают динамику работы этого элемента.

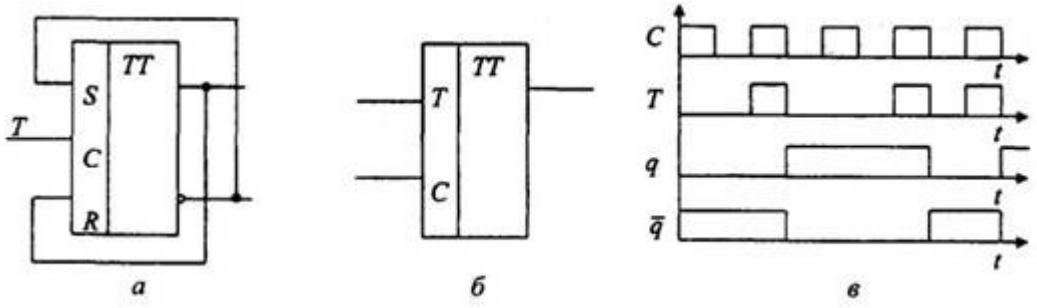


Рисунок 3.19 – Схема триггера со счетным входом:
а- функциональная; *б* - условное обозначение; *в* - временная диаграмма

Таблица 3.6 – Таблица переходов Т-триггера

Входные сигналы	Состояние q_t		Режим
X_t	0	1	
0	0	1	Хранение
1	1	0	Инверсия

По таблице переходов можно получить логическую функцию, реализуемую Т-триггером:

$$q_{t+1} = \overline{q}_t \cdot x_t \vee q_t \cdot \overline{x}_t = q_t \oplus x_t. \qquad (3.3)$$

Нетрудно видеть, что зависимость (3.3) очень похожа на функцию, выведенную для одноразрядного комбинационного полусумматора. На рисунке 3.19, *а*) показано, как двухтактный RS-триггер преобразуется в Т-триггер.

JK-триггер

Наиболее сложным типом триггера является **JK-триггер**. Он, по существу, является объединением двухтактного RS- и Т-триггеров. Этому соответствует его таблица переходов (таблица 3.7).

Таблица 3.7 – Таблица переходов JK-триггера

Входные сигналы		Состояние q		Режим
J	K	0	1	
0	1	0	1	Хранение
0	0	1	0	Установка 0
0	1	0	1	Установка 1
1	1	0	0	Инверсия

Если первые три строки таблицы переходов полностью повторяют соответствующие строки таблицы 3.4, то последняя строка, с запрещенной комбинацией для RS-триггера, соответствует режиму переключения Т-триггера (см. таблицу 3.6). Схема JK-триггера изображена на рисунке 3.20.

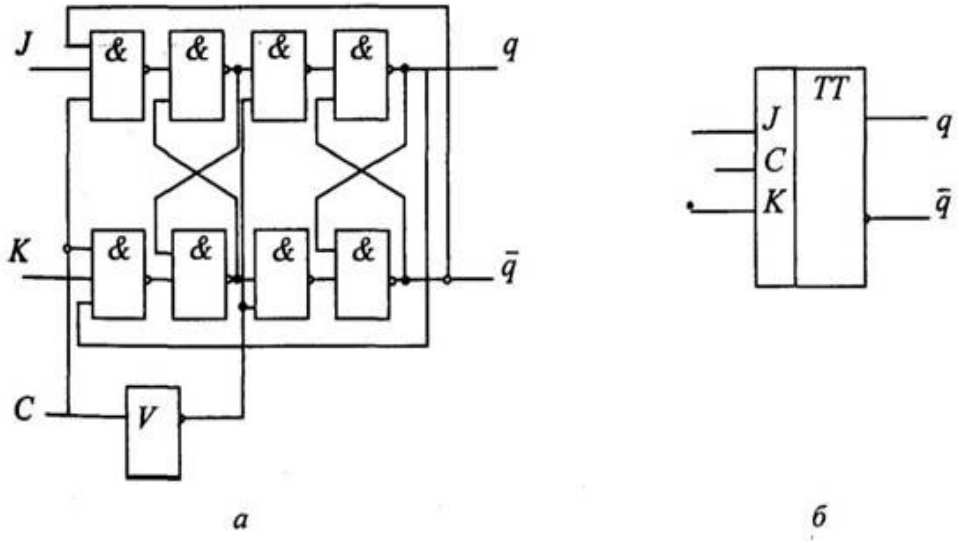


Рисунок 3.20 – JK-триггер:

а - функциональная схема; *б* - условное обозначение

На рисунке 3.21 приведена схема *двухступенчатого JK-триггера*. Следует отметить, что первая ступень представляет собой одноступенчатый *триггер*, реализующий заданную таблицу переходов, в то время как вторая ступень – это всегда одноступенчатый синхронный *RS-триггер*.

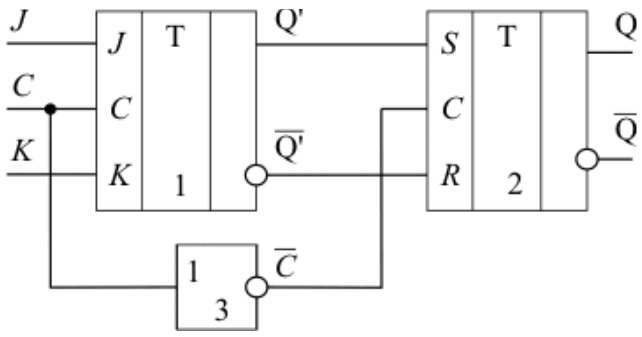


Рисунок 3.21 – Двухступенчатый JK-триггер

D-триггер

Динамический триггер D-триггер обычно строится на основе двухтактного RS- или JK-триггера. Он предназначается для хранения состояния (1 или 0) на один период тактовых импульсов (с задержкой на 1 такт). Таблица его переходов отражена в таблице 3.8. На рисунке 3.22, *а и б* представлены варианты его построения, а на рисунке 3.22, *в* - его условное обозначение.

Таблица 3.8 – Таблица переходов D-триггера

Входные сигналы	Состояния q		Режим
D	0	1	
0	0	0	Установка 0
1	1	1	Установка 1

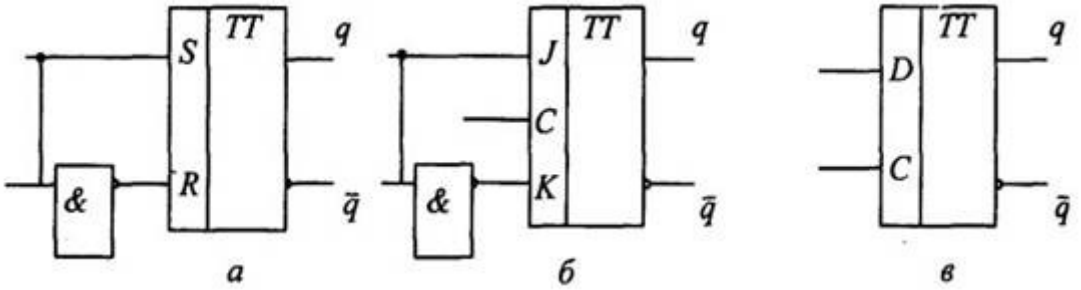


Рисунок 3.22 – Схема D-триггера:
а- функциональная схема на основе RS-триггера;
б- функциональная схема на основе JK-триггера;
в - условное обозначение

Все перечисленные элементы памяти позволяют хранить одну единицу информации – бит или одну двоичную цифру.

3.3 Узлы ЭВМ

При построении ЭВМ широко используются функциональные схемы, обеспечивающие операции хранения и преобразования информации над группами битов (машинными словами). Такие сложные схемы называются ***узлами***. К типовым узлам относят:

- регистры,
- счетчики,
- сумматоры.

Все они также принадлежат к регулярным структурам, состоящим из одинаковых параллельно работающих одноразрядных схем.

Регистры

Регистром называется узел, предназначенный для приема, временного хранения и выдачи машинного слова. Регистры могут также использоваться для некоторых операций преобразования данных:

- для сдвига кода числа (слова) на определенное число разрядов влево или вправо,
- для преобразования последовательного кода числа в параллельный и наоборот;
- для подсчета синхроимпульсов и т.д.

Эти дополнительные функции регистров обеспечиваются путем усложнения схем хранения, выбора более сложных триггеров и подключения дополнительных логических схем на их входах и выходах.

> Таким образом, **регистры** представляют собой *совокупность триггеров*, число которых соответствует числу разрядов в слове, *и вспомогательных схем*, обеспечивающих выполнение различных операций над словом.

На рисунке 3.23 показана функциональная схема *n*-разрядного регистра, построенного на RS-триггерах. Информация в регистр записывается под действием сигнала «Запись». Предварительно перед установкой кода на регистр обычно на все разряды R подается сигнал сброса. На рисунке 3.23 показано, что подключение к входам R дополнительных инверторов позволяет избежать этой предварительной операции. Здесь на вход каждого разряда поступает парафазный код двоичной цифры (x_i - на вход S_i и x_i - на вход R_i), т.е. прямое и инверсное значения кода подаются в противофазе.

На рисунке 3.24 изображена функциональная схема того же регистра, дополненная логическими элементами для преобразования хранящегося на регистре кода. По сигналу «Прямой код» с регистра считывается прямой код хранящихся данных, а по сигналу «Обратный код» - инверсное значение каждого разряда слова. Если оба эти сигнала поступают одновременно, то считывается парафазный код хранящейся информации. Более сложная логика на входе и выходе запоминающих элементов позволяет строить сдвигающие регистры.

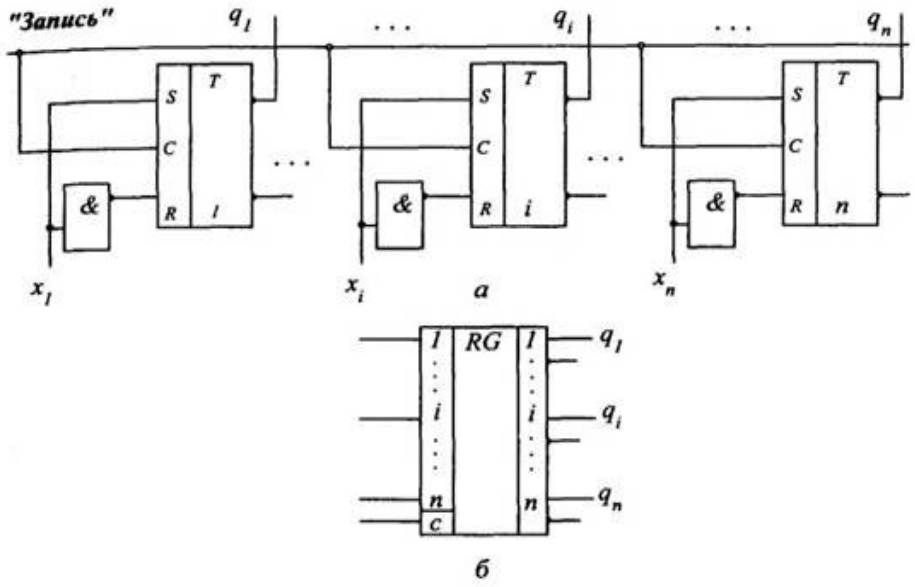


Рисунок 3.23 – Схема регистра на RS-триггерах:
а - функциональная схема; *б* - условное обозначение регистра

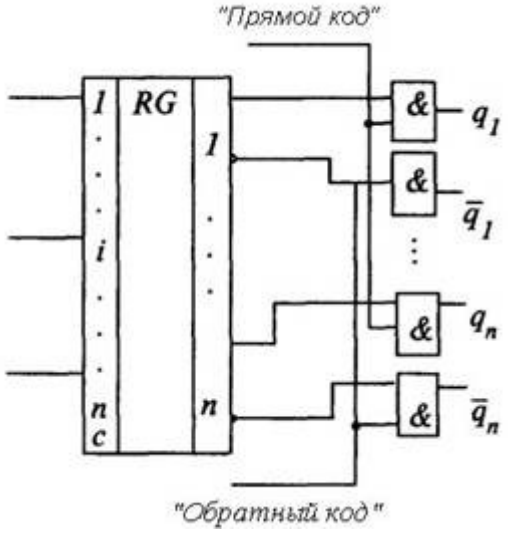


Рисунок 3.24 – Схема выдачи информации из регистра

Регистр хранения

Регистр хранения – внутреннее запоминающее устройство процессора или внешнего устройства, предназначенное для временного хранения обрабатываемой или управляющей информации.

Регистры представляют собой совокупность *триггеров*, количество которых равняется разрядности *регистра*, и вспомогательных схем, обеспечивающих выполнение некоторых элементарных операций. Набор этих операций, в зависимости от функционального назначения *регистра*, может включать в себя одновременную установку всех разрядов *регистра* в «0», параллельную или последовательную загрузку *регистра*, сдвиг содержимого *регистра* влево или вправо на требуемое число разрядов,

управляемую выдачу информации из *регистра* (обычно используется при работе нескольких схем на общую шину данных) и т.д.

Регистры хранения используются для приема, хранения и выдачи ***многоразрядного кода***. Они представляют собой совокупность одноступенчатых триггеров (как правило, *D*-типа) с общим входом синхронизации. Иногда в регистре имеется также и общий вход асинхронной установки всех триггеров в «0». Схема четырехразрядного *регистра хранения* приведена на рисунке 3.25, а его условно-графическое обозначение – на рисунке 3.26.

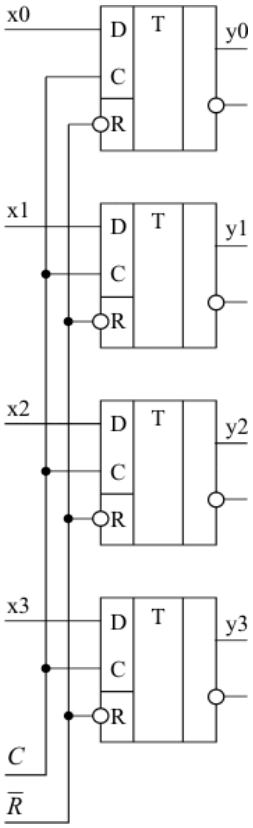


Рисунок 3.25 – Структура четырехразрядного регистра хранения с асинхронным входом установки в «0»

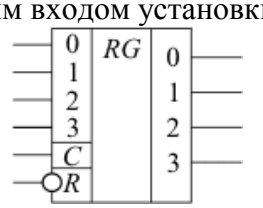


Рисунок 3.26 – Условно-графическое обозначение четырехразрядного регистра хранения с асинхронным входом установки в «0»

Регистр сдвига

Регистр сдвига – регистр, обеспечивающий помимо хранения информации, *сдвиг влево* или *вправо* всех разрядов одновременно на одинаковое число позиций. При этом выдвигаемые за пределы регистра разряды теряются, а в освобождающиеся разряды заносится информация, поступающая по отдельному внешнему входу регистра сдвиг*а*.

Обычно эти регистры обеспечивают сдвиг кода на одну позицию влево или вправо. Но существуют и универсальные регистры сдвига, которые выполняют сдвиг как влево, так и вправо в зависимости от значения сигнала на специальном управляющем входе или при подаче синхросигналов на разные входы регистра. Регистр сдвига может быть спроектирован и таким образом, чтобы выполнять сдвиг одновременно не на одну, а на несколько позиций.

Регистры сдвига строятся на *двухступенчатых триггерах*. Схема четырехразрядного регистра, выполняющего сдвиг на один разряд от разряда 0 к разряду 3, показана на рисунке 3.28, а его условно-графическое обозначение – на рисунке 3.27. Ввод информации в данный регистр – последовательный через внешний вход D_0. Регистр имеет вход асинхронной установки всех разрядов в «0». Для наглядности каждый двухступечатый регистр представлен двумя одноступенчатыми с соответствующей организацией синхронизации первой и второй ступеней. Пунктиром обозначен реальный двухступенчатый триггер.

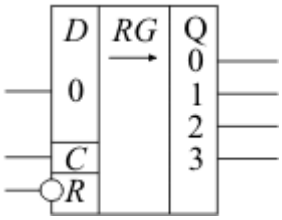


Рисунок 3.27 – Условно-графическое обозначение четырехразрядного регистра сдвига с асинхронным входом установки в «0»

Идеализированная временная диаграмма работы регистра сдвига показана на рисунке 3.29. Предполагаем, что начальное состояние *регистра* следующее: $Q_0=0$, $Q_1=1$, $Q_2=1$, $Q_3=0$.

Работа регистра сдвига в каждом периоде сигнала синхронизации разбивается на две фазы: при высоком и при низком значении синхросигнала:

- При высоком уровне синхросигнала проводится запись значения выхода (i–1)-го разряда *регистра* в первую ступень i-го разряда. Вторая ступень каждого разряда сохраняет свое прежнее значение. В этой фазе состояние первой ступени i-го *триггера* повторяет состояние второй ступени (i – 1)-го *триггера*. Вторые ступени каждого *триггера*, а, следовательно, и выходы *регистра* в целом, остаются неизменными.
- При низком уровне синхросигнала значение, записанное в первой ступени каждого *триггера*, перезаписывается в его вторую ступень. Запись в первую ступень

триггера запрещена. В этой фазе состояния первой и второй ступеней каждого *триггера* становятся одинаковыми.

➢ Поступление сигнала R = 0 вне зависимости от значения сигнала на входе синхронизации C и сигнала на входе D0 устанавливает все разряды *регистра* в нулевое состояние.

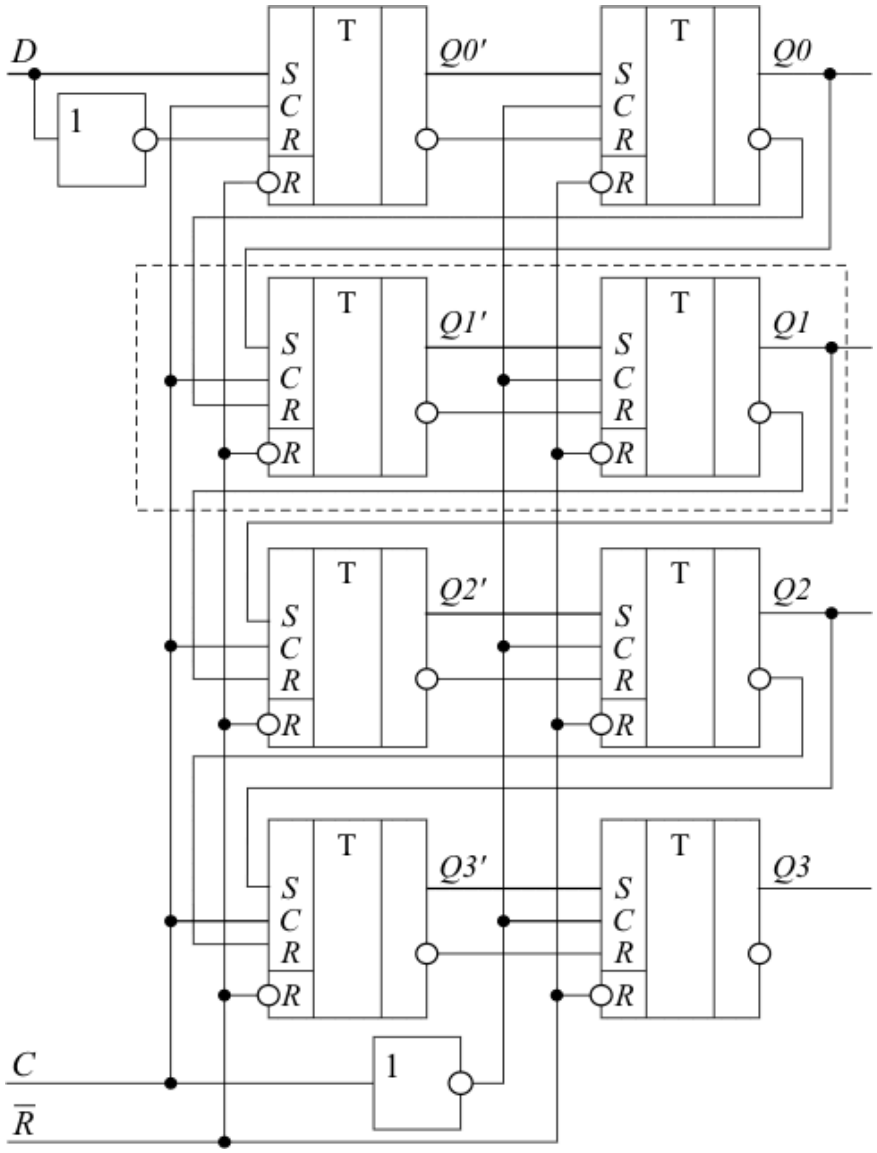


Рисунок 3.28 – Структура регистра сдвига

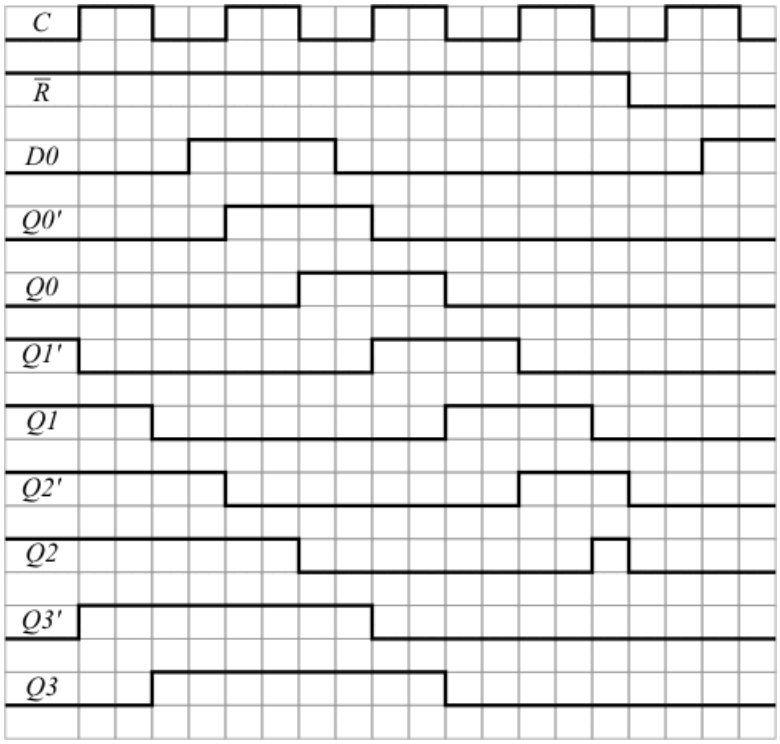


Рисунок 3.29 – Временная диаграмма работы регистра сдвига

Счетчик

Счетчик – узел ЭВМ, позволяющий осуществлять подсчет поступающих на его вход сигналов и фиксацию результата в виде многоразрядного двоичного числа. Счетчик, состоящий из n-триггеров, дает возможность подсчитывать до N сигналов, связанных зависимостью:

$n = \log 2\ N$ или $N = 2^n$.

В ЭВМ счетчики используются для подсчета импульсов, сдвигов, формирования адресов и т.д. Функционально различают:

- суммирующие,
- вычитающие,
- реверсивные счетчики.

Они также отличаются друг от друга логикой работы дополнительных логических элементов, подключаемых к триггерам.

В основу построения любого счетчика положено свойство Т-триггеров изменять свое состояние при подаче очередного сигнала на счетный вход Т. На рисунке 3.30 показана схема трех разрядов суммирующего счетчика, построенного на Т-триггерах. Логика его работы представлена в таблице 3.9.

Таблица 3.9 – Таблица переходов трехразрядного счетчика

Вход	Состояние								
x	000	001	010	011	100	101	110	111	Режим
0	000	001	010	011	100	101	110	111	Хранение
1	001	010	011	100	101	110	111	000	Счет

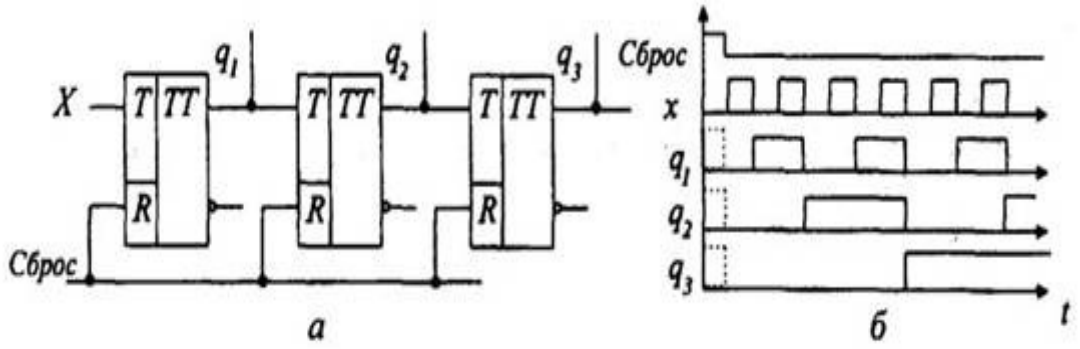


Рисунок 3.30 – Организация счетчика на Т-триггерах:
а - функциональная схема; *б* - временная диаграмма

Сумматор

Сумматор - узел ЭВМ, в котором суммируются коды чисел. Как правило, любой сумматор представляет собой комбинацию одноразрядных сумматоров. Сумматоры различают по принципам построения:

- накапливающего типа
- комбинационного типа.

Сумматоры накапливающего типа строят на сложных JK и/или RS-триггерах, дополняя их выходы достаточно сложными схемами формирования и распространения переносов. Процесс сложения при этом осуществляется поэтапно. Сначала на триггерах сумматора фиксируется код первого операнда, затем на счетные коды разрядов подается код второго операнда. На каждом триггере формируются одноразрядные суммы и значения переносов между разрядами. Учет возникающих переносов задерживает формирование окончательного результата суммы и может требовать дополнительных тактов сложения. Из-за этого многоразрядные схемы сумматора накапливающего типа используются достаточно редко.

Более часто для построения сумматоров используются сумматоры комбинационного типа. Логика работы такого сумматора была представлена данными таблице 2.2. Обычно у такого сумматора на входе и выходе имеются регистры для хранения и преобразования кодов операндов и результата (рисунок 3.31).

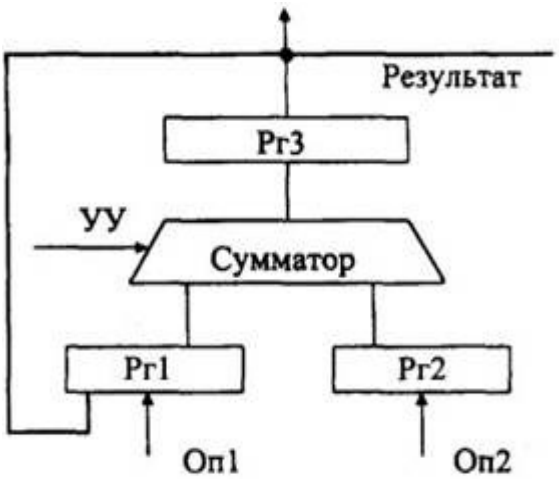


Рисунок 3.31 – Упрощенная схема сумматора ЭВМ

Регистр Рг1 предназначается для хранения кода первого операнда, регистр Рг2 - для хранения кода второго операнда. Сумматор по сигналам из устройства управления настраивается на выполнение определенной машинной операции, соответствующей коду операции, находящемуся в коде команды. Результат выполняемой операции фиксируется в регистре РгЗ. При необходимости этот результат может использоваться для продолжения вычислений. Для этого предусматривается возможность перезаписи содержимого регистра РгЗ на Рг1 в качестве значения одного из операндов при выполнении очередной операции.

Арифметико-логическое устройство (АЛУ)

Классическая ЭВМ состоит из трех основных устройств:

- арифметико-логического устройства,
- устройства управления
- запоминающего устройства.

Рассмотрим особенности организации *арифметико-логического устройства*.

В современных ЭВМ *АЛУ* не является самостоятельным схемотехническим блоком. Оно входит в состав микропроцессора, на котором строится компьютер. Однако знание структуры и принципов работы *АЛУ* весьма важно для понимания работы компьютера в целом. Для лучшего понимания этих вопросов проведем синтез арифметического устройства, предназначенного для выполнения только одной операции – *умножения* чисел с фиксированной запятой, заданных в прямом коде, со старших разрядов множителя. В ходе этого процесса также обратим внимание на особенности использования рассмотренных выше основных схемотехнических элементов ЭВМ.

Синтез *АЛУ* проходит в несколько этапов. Сначала необходимо выбрать метод, по которому предполагается выполнение операции, и составить алгоритм соответствующих

действий. Исходя из алгоритма и формата исходных данных, следует определить набор составляющих *АЛУ* элементов. Затем требуется определить связи между элементами, установить порядок функционирования устройства и временную диаграмму управляющих сигналов, которые должны быть поданы на *АЛУ* от устройства управления.

Пусть операнды имеют вид:

$[X]_{пк} = x_0x_1x_2\ldots x_n$

$[Y]_{пк} = y_0y_1y_2\ldots y_n$

где x_0, y_0 – знаковые разряды.

Операция *умножения* чисел с фиксированной запятой, заданных в прямом коде, со старших разрядов множителя выполняется по следующей формуле:

Sign Z = Sign X $\oplus$ Sign Y

$|Z| = y_1 \bullet |X| \bullet 2^{-1} + y_2 \bullet |X| \bullet 2^{-2} + \ldots + y_n \bullet |X| \bullet 2^{-n}$

$[X]_{пк}$ = 0.1101 ; Sign X = 0

$[Y]_{пк}$ = 1.1011 ; Sign Y = 1

Sign Z = 0 $\oplus$ 1 = 1

|X| = 0. 1 1 0 1

|Y| = 0. 1 0 1 1

$y_1y_2y_3y_4$

+0.00000000 |Z| = 0

y_1 = 1 0.01101000 $1 \bullet |X| \bullet 2^{-1}$

+0.01101000 $|Z| = |Z| + |X| \bullet 2^{-1}$

y_2 = 0 0.00000000 $0 * |X| \bullet 2^{-2}$

+0.01101000 |Z| = |Z| + 0

y_3 = 1 0.00011010 $1 \bullet |X| \bullet 2^{-3}$

+0.10000010 $|Z| = |Z| + |X| \bullet 2^{-3}$

y_4 = 1 0.00001101 $1 \bullet |X| \bullet 2^{-4}$

0.10001111 $|Z| = |Z| + |X| \bullet 2^{-4}$

Алгоритм вычислений представлен на рисунке 3.32.

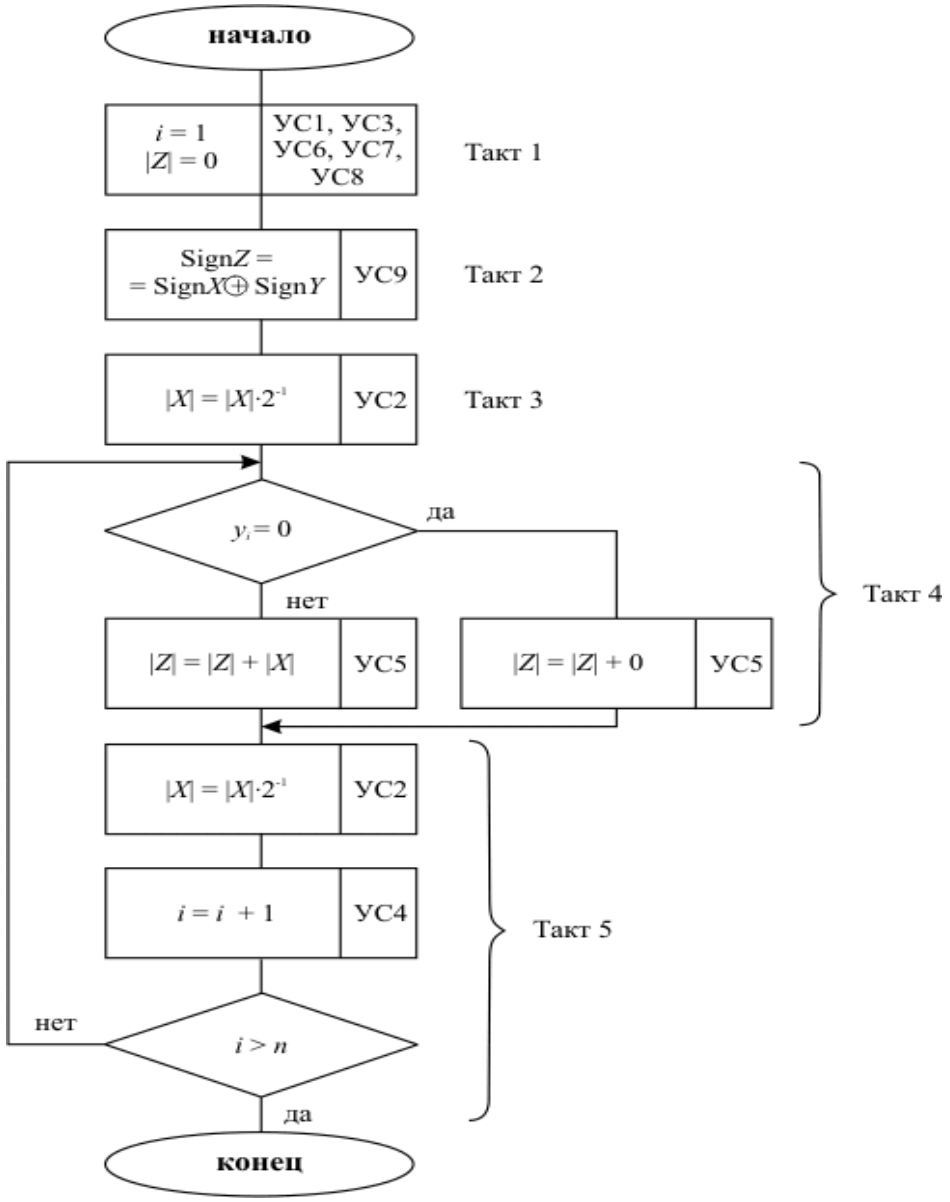


Рисунок 3.32 – Алгоритм операции умножения чисел с фиксированной запятой, заданных в прямом коде, со старших разрядов множителя

Каждой переменной, представленной в алгоритме, в схеме должен соответствовать элемент хранения. Разрядность модуля произведения равна сумме разрядностей сомножителей. *Умножение* двоичного числа на 2^{-i} обеспечивается сдвигом этого числа вправо на соответствующее количество разрядов. Переход к анализу очередного разряда множителя (`i = i + 1`) может быть обеспечен сдвигом регистра множителя на один разряд в сторону старших разрядов.

Исходя из этого, определим состав оборудования, необходимого для реализации *АЛУ* заданного типа для `n = 4` (таблица 3.10).

Таблица 3.10 – Состав оборудования для АЛУ

Схема	Разрядность	Функции	Управляющий сигнал
Регистр модуля множимого RGX	8	Загрузка. Сдвиг в сторону младших разрядов.	УС1 УС2
Регистр модуля множителя RGY	4	Загрузка. Сдвиг в сторону старших разрядов.	УС3 УС4
Регистр модуля результата RGZ	8	Загрузка. Установка в "0".	УС5 УС6
Триггер знака множимого TX		Загрузка	УС7
Триггер знака множителя TY		Загрузка	УС8
Триггер знака результата TZ		Загрузка	УС9
АЛУ	8	Комбинационный *сумматор*	–
Комбинационные схемы		Получение на входе *АЛУ* сигналов "0" или RGX в зависимости от значения y_i	–

Структурная схема устройства представлена на рисунке 3.33. Временная диаграмма управляющих сигналов, поступающих на *арифметико-логическое устройство*, показана на рисунке 3.34.

Работа схемы

Такт 1. Загрузка модулей операндов в регистры RGX, RGY, а их знаков – в триггеры TX и TY. Сброс в "0" регистра результата RGZ.

Такт 2. Запись знака результата в триггер TZ.

Такт 3. Сдвиг регистра RGX на один разряд вправо. Через время, равное задержке на переключение регистров и *комбинационных схем*, на выходе комбинационного *сумматора* и, следовательно, на входе регистра RGZ устанавливается результат $0+y_1 \cdot |X| \cdot 2^{-1}$.

Такт 4. Загрузка RGZ: $|Z|=|Z|+y_1 \cdot |X| \cdot 2^{-1}$.

Такт 5. Сдвиг RGX на 1 разряд вправо: $|X| = |X| \cdot 2^{-1}$.

Сдвиг RGY на 1 разряд влево: i=i+1.

Устройство управления проверяет условие окончания операции: i > n.

Такты (6,7), (8,9), (10,11) ... Повтор действий тактов (4,5) с анализом других значений y_i. В такте 10 в регистре RGZ формируется модуль произведения. Такт 11 используется лишь для определения условия окончания операции *умножения*.

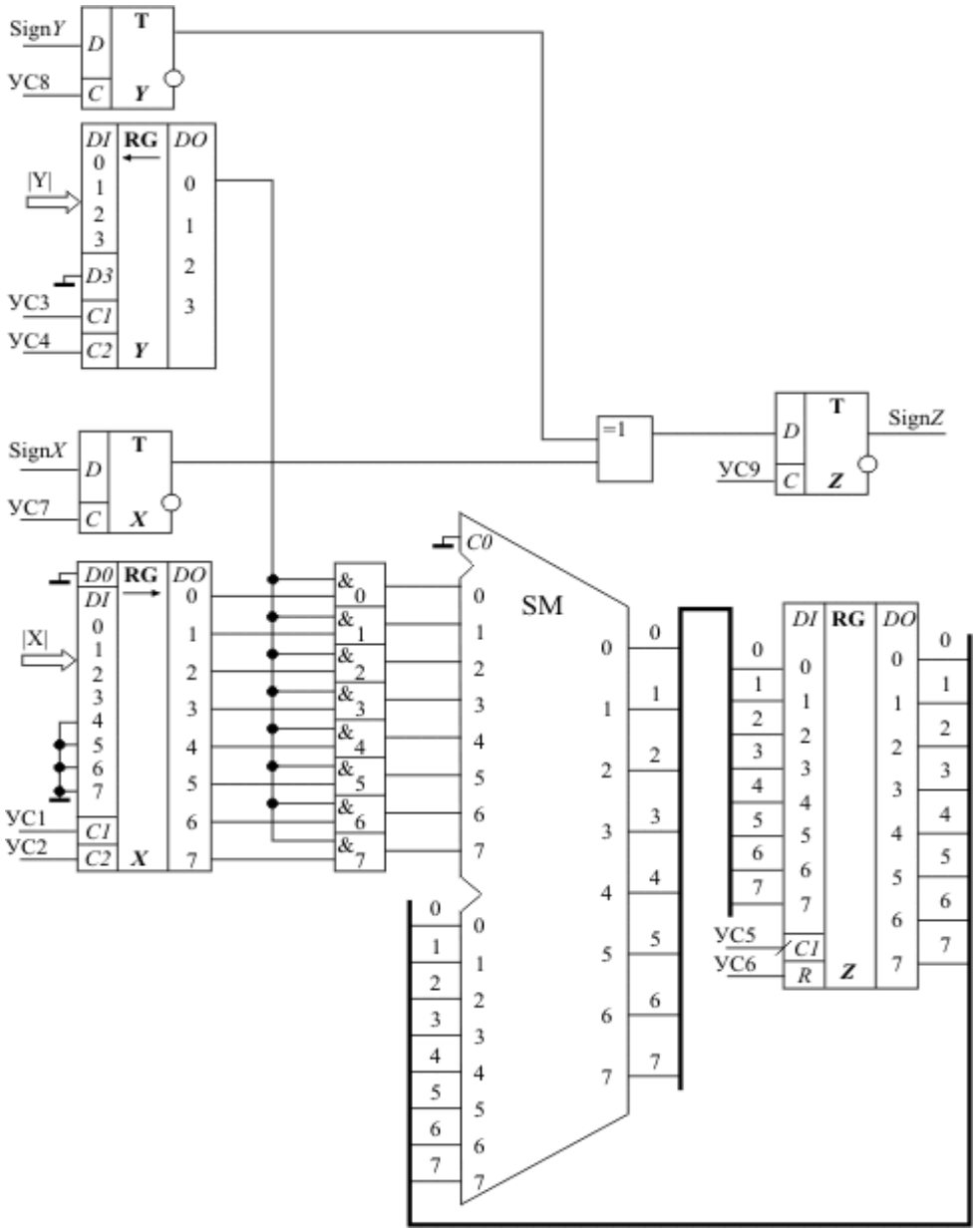


Рисунок 3.33 – Структурная схема АЛУ для выполнения операции умножения со старших разрядов множителя чисел, заданных в прямом коде

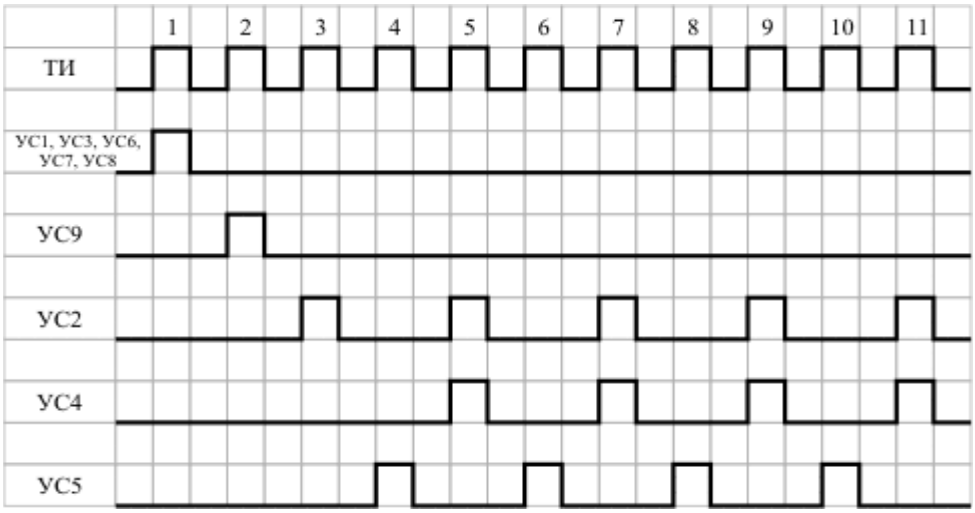


Рисунок 3.34 – Временная диаграмма управляющих сигналов

Устройство управления

Компьютер условно можно разделить на два основных блока:

- Операционный
- Управляющий.

Для реализации любой команды необходимо на соответствующие управляющие входы любого устройства компьютера подать определенным образом распределенную во времени последовательность управляющих сигналов. Часть цифрового вычислительного устройства, предназначенная для выработки этой последовательности, называется ***устройством управления (УУ)***.

Любое действие, выполняемое в операционном блоке, описывается некоторой ***микропрограммой*** и реализуется за один или несколько тактов.

> Элементарная функциональная операция, выполняемая за один тактовый интервал и приводимая в действие управляющим сигналом, называется ***микрооперацией.***

Например, в спроектированном АЛУ для умножения чисел в первом такте выполняются следующие *микрооперации*: `TX=0`, `TY=0`, `RGX=|X|`, `RGY=|Y|`, `RGZ=0`.

> Совокупность *микроопераций*, выполняемых в одном такте, называется ***микрокомандой*** (МК).

Если все такты должны иметь одну и ту же длину, а именно это имеет место при работе компьютера, то она устанавливается по самой продолжительной *микрооперации*.

> *Микрокоманды*, предназначенные для выполнения некоторой функционально законченной последовательности действий, образуют ***микропрограмму***.

Например, *микропрограмму* образует набор *микрокоманд* для выполнения команды умножения.

Устройство управления предназначено для выработки управляющих сигналов, под воздействием которых происходит преобразование информации в арифметико-логическом устройстве, а также операции по записи и чтению информации в/из запоминающего устройства.

Устройства управления делятся на:

- УУ с жесткой, или схемной логикой и
- УУ с программируемой логикой (*микропрограммные УУ*).

В *устройствах управления* первого типа для каждой команды, задаваемой кодом операции, строится набор комбинационных схем, которые в нужных тактах вырабатывают необходимые управляющие сигналы.

В микропрограммных УУ каждой команде ставится в соответствие совокупность хранимых в специальной памяти слов - *микрокоманд*. Каждая из *микрокоманд* содержит информацию о *микрооперациях*, подлежащих выполнению в данном такте, и указание, какое слово должно быть выбрано из памяти в следующем такте.

Схемное устройство управления

Устройство управления схемного типа (см. рисунок 3.35) состоит из:

- ***датчика сигналов***, вырабатывающего последовательность импульсов, равномерно распределенную во времени по своим шинам (рисунок 3.36) (n - общее количество управляющих сигналов, необходимых для выполнения любой операции; m - количество тактов, за которое выполняется самая длинная операция);
- ***блока управления операциями***, осуществляющего выработку управляющих сигналов, то есть коммутацию сигналов, поступающих с ДС, в соответствующем такте на нужную управляющую шину;
- ***дешифратора кода операций***, который дешифрирует код операции команды, присутствующей в данный момент в регистре команд, и возбуждает одну шину, соответствующую данной операции; этот сигнал используется блоком управления операциями для выработки нужной последовательности управляющих сигналов.

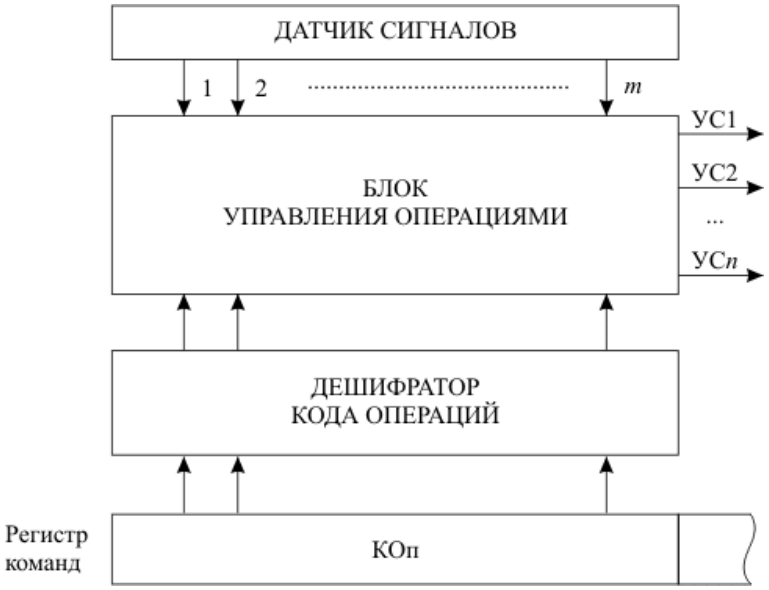


Рисунок 3.35 – Функциональная схема схемного устройства управления

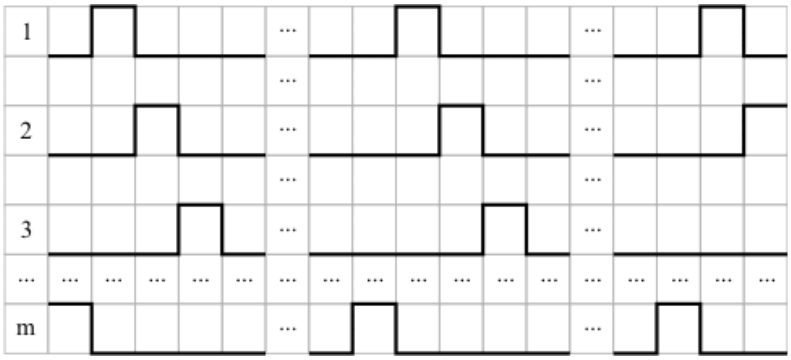


Рисунок 3.36 – Временная диаграмма работы датчика сигналов

Датчик сигналов обычно реализуется на основе счетчика с дешифратором или на сдвиговом регистре.

Датчик сигналов на основе счетчика с дешифратором

Реализация датчика сигналов на основе счетчика с дешифратором представлена на рисунке 3.37. По заднему фронту каждого тактового импульса, поступающего на устройство управления с системного генератора импульсов, счетчик увеличивает свое состояние; выходы счетчика соединены со входами дешифратора, выходы которого и являются выходами датчика сигналов рисунке 3.38.

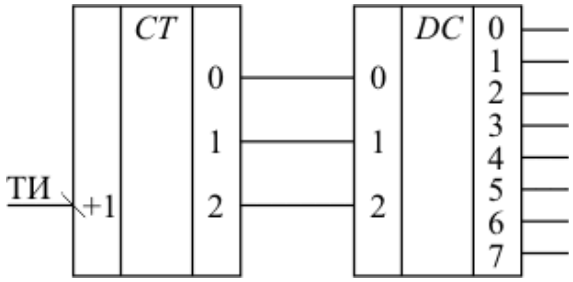


Рисунок 3.37 – Схема датчика сигналов на основе счетчика с дешифратором

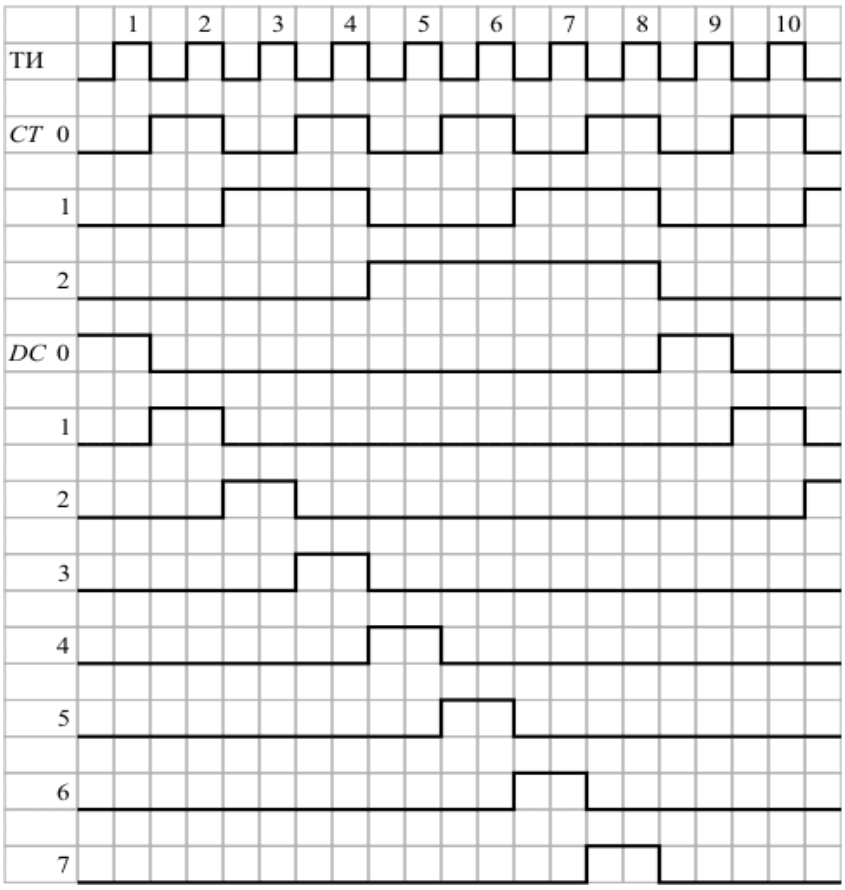


Рисунок 3.38 – Временная диаграмма работы датчика сигналов на основе счетчика с дешифратором

Датчик сигналов на сдвиговом регистре

Проектирование *датчика сигналов* на сдвиговом регистре требует лишь его "закольцовывания", то есть соединения выхода последнего разряда с входом, через который в регистр заносится информация при сдвиге, и первоначальной установки (рисунок 3.39). В начальном состоянии регистр содержит «1» только в разряде 0. Входы параллельной загрузки регистра для его начальной установки и соответствующий этой операции управляющий вход регистра на схеме не показаны.

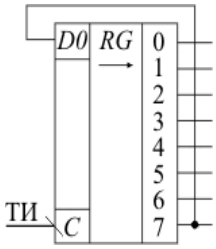


Рисунок 3.39 – Схема датчика сигналов на основе регистра сдвига

Временная диаграмма работы этой схемы приведена на рисунке 3.40.

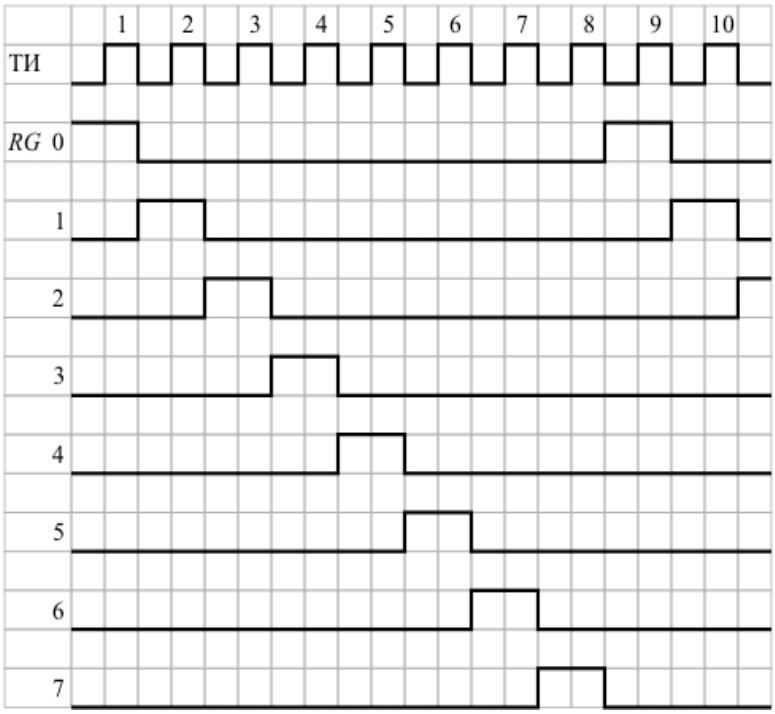


Рисунок 3.40 – Временная диаграмма работы датчика сигналов на основе регистра сдвига

Наиболее сложной частью *схемного устройства управления* является блок управления операциями. Он представляет собой нерегулярную схему, структура которой определяется системой команд и составом оборудования процессора. Такое УУ может быть реализовано в виде специализированной интегральной схемы.

Структурная схема микропрограммного устройства управления

Микропрограммное устройство управления представлено на рисунке 3.41. Преобразователь адреса *микрокоманды* преобразует код операции команды, присутствующей в данный момент в регистре команд, в начальный адрес *микропрограммы*, реализующей данную операцию, а также определяет адрес следующей *микрокоманды* выполняемой *микропрограммы* по значению адресной части текущей *микрокоманды*.

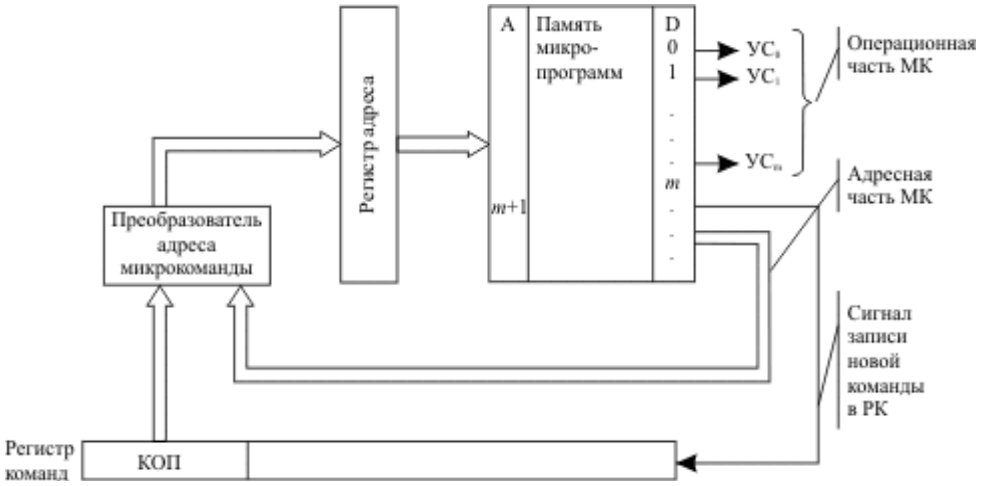


Рисунок 3.41 – Функциональная схема микропрограммного устройства управления (УСi - управляющие сигналы, вырабатываемые устройством управления)

В таблице 3.11 приведен пример *микропрограммы* для выполнения операции умножения чисел в дополнительном коде. Предполагается, что начальный адрес *микропрограммы* равен 300, количество разрядов множителя равно 2, а адресная часть *микрокоманды* содержит адрес *микрокоманды*, которая должна быть выбрана в следующем такте. В последней *микрокоманде* в регистр команд загрузится очередная команда, код операции которой определит начальный адрес очередной *микропрограммы*. В реальных *микропрограммных устройствах управления* формирование адреса следующей *микрокоманды* проводится более сложным образом, учитывающим возможности ветвлений и циклического повторения отдельных фрагментов *микропрограмм*.

Таблица 3.11 – Микропрограмма выполнения операции умножения

Адрес МК	УС1	УС2	УС3	УС4	УС5	УС6	Сигнал записи в РК	Адрес следующей МК
300	1	0	1	0	0	1	0	301
301	0	0	0	0	1	0	0	302
302	0	1	0	1	0	0	0	303
303	0	0	0	0	1	0	0	304
304	0	1	0	1	0	0	1	X

Из анализа структуры и принципов работы *схемного* и *микропрограммного устройств управления* видно, что УУ первого типа имеют сложную нерегулярную структуру, которая требует специальной разработки для каждой системы команд и должна практически полностью перерабатываться при любых модификациях системы команд. В то же время оно имеет достаточно высокое быстродействие, определяемое быстродействием используемого элементного базиса.

Устройство управления, реализованное по микропрограммному принципу, может легко настраиваться на возможные изменения в операционной части ЭВМ. При этом настройка во многом сводится лишь к замене *микропрограммной памяти*. Однако УУ этого типа обладают худшими временными показателями по сравнению с *устройствами управления* на жесткой логике

3.5 Проблемы развития элементной базы

Одним из главных факторов достижения высокого быстродействия, а, значит, и высокой производительности ЭВМ является построение их на новейшей элементной базе. Смена поколений ЭВМ в значительной степени связана с переходами на новые поколения элементной базы, знаменующие достижения новых частотных диапазонов работы схем в рамках доступных технологий. Успехи в создании новой элементной базы определяются передовыми научными и техническими достижениями целого ряда наук (физики, химии, оптики, механики и др.). Качество элементной базы является показателем 'технического прогресса.

Все современные ЭВМ строятся на *микропроцессорных наборах*, основу которых составляют *большие (БИС) и сверхбольшие интегральные схемы (СБИС)*. Технологический принцип разработки и производства интегральных схем действует уже более четверти века. Он заключается в циклическом послойном изготовлении частей электронных схем по циклу программа - рисунок - схема. По программам на напыленный фоторезисторный слой наносится рисунок будущего слоя микросхемы. Затем рисунок протравливается, фиксируется, закрепляется и изолируется от новых слоев. На основе этого создается пространственная твердотельная структура. Например, СБИС типа Pentium включает около трех с половиной миллионов транзисторов, размещаемых в пятислойной структуре.

Степень микроминиатюризации, размер кристалла ИС, производительность и стоимость технологии напрямую определяются типом литографии. До настоящего времени доминирующей оставалась оптическая литография, т.е. послойные рисунки на фоторезисторе микросхем наносились световым лучом. В настоящее время ведущие компании, производящие микросхемы, реализуют кристаллы с размерами примерно 400мм**2** - для процессоров (например, Pentium) и 200мм² - для схем памяти. Минимальный топологический размер (толщина линий) при этом составляет 0,5 - 0,35 мкм. Для сравнения можно привести такой пример. Толщина человеческого волоса составляет примерно 100 мкм. Значит, при таком разрешении на толщине волоса могут вычерчивать более двухсот линий.

Дальнейшие достижения в микроэлектронике связываются с электронной (лазерной), ионной и рентгеновской литографией. Это позволяет выйти на размеры 0.25, 0.18 и даже 0.08мкм.

При таких высоких технологиях возникает целый ряд проблем. Микроскопическая толщина линий, сравнимая с диаметром молекул, требует высокой чистоты используемых и напыляемых материалов, применения вакуумных установок и снижения рабочих температур. Действительно, достаточно попадания мельчайшей пылинки при изготовлении микросхемы, как она попадает в брак. Поэтому новые заводы по производству микросхем имеют уникальное оборудование, размещаемое в чистых помещениях класса 1, микросхемы в которых транспортируются от оборудования к оборудованию в замкнутых сверхчистых мини-атмосферах класса 1000. Мини-атмосфера создается, например, сверхчистым азотом или другим инертным газом при давлении 10-4 Торр.

Уменьшение линейных размеров микросхем и повышение уровня их интеграции заставляют проектировщиков искать средства борьбы с потребляемой *Wn* и рассеиваемой *Wp* мощностью. При сокращении линейных размеров микросхем в 2 раза их объемы изменяются в 8 раз. Пропорционально этим цифрам должны меняться и значения *Wn* и *Wp,* в противном случае схемы будут перегреваться и выходить из строя. В настоящее время основой построения всех микросхем была и остается КМОП-технология (комплиментарные схемы, т.е. совместно использующие n- и p-переходы в транзисторах со структурой металл - окисел -полупроводник).

Известно, что *W=U*I.* Напряжение питания современных микросхем составляет 5 - 3V. Появились схемы с напряжением питания 2,8V, что выходит за рамки принятых стандартов. Дальнейшее понижение напряжения нежелательно, так как всегда в электронных схемах должно быть обеспечено необходимое соотношение сигнал-шум, гарантирующее устойчивую работу ЭВМ.

Протекание тока по микроскопическим проводникам сопряжено с выделением большого количества тепла. Поэтому, создавая сверхбольшие интегральные схемы, проектировщики вынуждены снижать тактовую частоту работы микросхем. На рисунке 3.42 показано, что использование максимальных частот работы возможно только в микросхемах малой и средней интеграции. Максимальная частота *f*max=1011-1012Гц доступна очень немногим материалам: кремнию Si, арсениду галлия GaAs и некоторым другим. Поэтому они чаще всего и используются в качестве подложек в микросхемах.

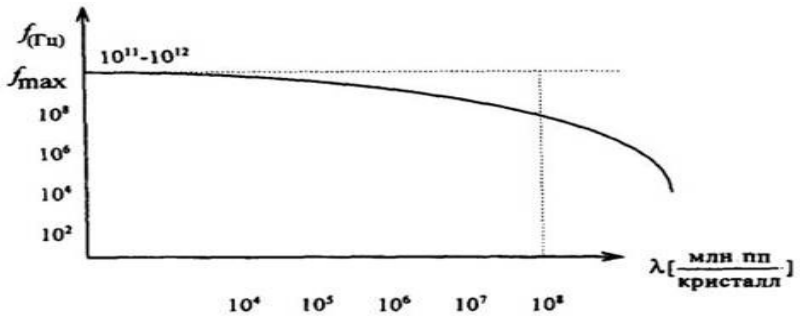


Рисунок 3.42 –Зависимость частоты *f* от степени интеграции λ

Таким образом, переход к конструированию ЭВМ на СБИС и ультра-СБИС должен сопровождаться снижением тактовой частоты работы схемы. Дальнейший прогресс в повышении производительности может быть обеспечен либо за счет архитектурных решений, либо за счет новых принципов построения и работы микросхем. Альтернативных путей развития просматривается не очень много. Так как микросхемы СБИС не могут работать с высокой тактовой частотой, то в ЭВМ будущих поколений их целесообразно комплексировать в системы. При этом несколько СБИС должны работать параллельно, а слияние работ в системе должно обеспечивать сверхскоростные ИС (ССИС), которые не могут иметь высокой степени интеграции.

Большие исследования проводятся также в области использования явления сверхпроводимости и туннельного эффекта - эффекта Джозефсона. Работа микросхем при температурах, близких к абсолютному нулю (-273°C), позволяет достигнуть *f*max, при этом *Wp=Wn=0.* Очень интересны результаты по использованию "теплой сверхпроводимости". Оказывается, что для некоторых материалов, в частности для солей бария, явление сверхпроводимости наступает уже при температурах около –1500С. Высказывались соображения, что могут быть получены материалы, имеющие сверхпроводимость при температурах, близких к комнатной. С уверенностью можно сказать, что появление таких элементов знаменовало бы революцию в развитии средств вычислительной техники новых поколений.

В качестве еще одного из альтернативных путей развития элементной *базы* ЭВМ будущих поколений следует рассматривать и биомолекулярную технологию. В настоящее время имеются опыты по синтезу молекул на основе их стереохимического генетического кода, способных менять ориентацию и реагировать на ток, на свет и т.п. Однако построение из них биологических микромашин еще находится на стадии экспериментов. Таким образом, можно сделать вывод, что в настоящее время возможности микроэлектроники еще не исчерпаны, но давление пределов уже ощутимо. Основой для

ЭВМ будущих поколений будут БИС и СБИС совместно с ССИС. При этом структуры ЭВМ и ВС будут широко использовать параллельную работу микропроцессоров.

ГЛАВА 4 ФУНКЦИОНАЛЬНАЯ И СТРУКТУРНАЯ ОРГАНИЗАЦИЯ ЭВМ

4.1 Общие принципы функциональной и структурной организации ЭВМ

Электронные вычислительные машины включают, кроме аппаратурной части и программного обеспечения (ПО), большое количество функциональных средств. К ним относятся коды, с помощью которых обрабатываемая информация представляется в цифровом виде:

- ***арифметические коды*** - для выполнения арифметических преобразований числовой информации;
- ***помехозащитные коды***, используемые для защиты информации от искажений;
- ***коды формы***, определяющие, как должна выглядеть обрабатываемая в ЭВМ информация при отображении;
- ***цифровые коды аналоговых величин*** (звука, «живого видео») и др.

Кроме кодов на функционирование ЭВМ оказывают влияние *алгоритмы* их формирования и обработки, *технология выполнения различных процедур* (например, начальной загрузки операционной системы, принятой в системе технологии обработки заданий пользователей и др.); *способы использования различных устройств и организация их работы* (например, организация системы прерываний или организация прямого доступа к памяти), устранение негативных явлений (например, таких, как фрагментация памяти) и др.

> Будем считать, что коды, система команд, алгоритмы выполнения машинных операций, технология выполнения различных процедур и взаимодействия аппаратного и программного обеспечений, способы использования устройств при организации их совместной работы, составляющие идеологию функционирования ЭВМ, образуют ***функциональную организацию ЭВМ***.

Реализована идеология функционирования ЭВМ может быть по-разному: аппаратными, программно-аппаратными или программными средствами. При *аппаратурной и программно-аппаратной реализации* могут быть применены регистры, дешифраторы, сумматоры; блоки жесткого аппаратурного управления или микропрограммного с управлением подпрограммами (комплексами микроопераций); устройства или комплексы устройств, реализованные в виде автономных систем (программируемых или с жестким управлением) и др. При *программной реализации* могут быть применены различные виды программ - обработчики прерываний, резидентные или загружаемые драйверы, com-, exe- или tsr - программы, bat- файлы и др.

Будем считать, что способы реализации функций ЭВМ составляют ***структурную организацию ЭВМ***.

Тогда элементная база, функциональные узлы и устройства ЭВМ, программные модули различных видов (обработчики прерываний, драйверы, com-, exe-, tsr-программы, bat-файлы и др.) являются структурными компонентами ЭВМ.

При серьезных конструктивных различиях ЭВМ могут быть совместимыми, т.е. приспособленными к работе с одними и теми же программами (*программная совместимость*) и получению одних и тех же результатов при обработке одной и той же, однотипно представленной информации (*информационная совместимость*). Если аппаратная часть электронных вычислительных машин допускает их электрическое соединение для совместной работы и предусматривает обмен одинаковыми последовательностями сигналов, то имеет место и *техническая совместимость* ЭВМ.

Совместимые ЭВМ должны иметь одинаковую функциональную организацию:

✦ информационные элементы (символы) должны одинаково представляться при вводе и выводе из ЭВМ,

✦ система команд должна обеспечивать в этих ЭВМ получение одинаковых результатов при одинаковых преобразованиях информации.

Работой таких машин должны управлять одинаковые или функционально совместимые операционные системы (а для этого должны быть совместимы методы и алгоритмы планирования и управления работой аппаратурно-программного вычислительного комплекса). Аппаратные средства должны иметь согласованные питающие напряжения, частотные параметры сигналов, а главное - состав, структуру и последовательность выработки управляющих сигналов.

При неполной совместимости ЭВМ (при наличии различий в их функциональной организации) применяют ***эмуляторы***, т.е. программные преобразователи функциональных элементов.

Состав функциональных блоков и структурных средств неоднороден. Поэтому в большинстве случаев функциональная и структурная организация будут рассматриваться в тех разделах, которые посвящены соответствующим аппаратурной части (hardware) или программному обеспечению (Software).

4.2 Организация функционирования ЭВМ с магистральной архитектурой

ЭВМ представляет собой совокупность устройств, выполненных на больших интегральных схемах, каждая из которых имеет свое функциональное назначение.

Комплект интегральных схем, из которых состоит ЭВМ, называется ***микропроцессорным комплектом***.

В состав микропроцессорных комплектов входят: системный таймер, микропроцессор (МП), сопроцессоры, контроллер прерываний, контроллер прямого доступа к памяти, контроллеры устройств ввода-вывода и др.

Все устройства ЭВМ делятся на *центральные* и *периферийные.* Центральные устройства полностью электронные, периферийные устройства могут быть либо электронными, либо электромеханическими с электронным управлением.

В центральных устройствах основным узлом, связывающим микропроцессорный комплект в единое целое, является ***системная магистраль (СМ)***. Она состоит из четырех узлов, называемых шинами:

- шина данных (ШД),
- шина адреса (ША),
- шина управления (ШУ),
- шина питания (ШП).

В состав системной магистрали входят *регистры-защелки*, в которых запоминается передаваемая информация, *шинные формирователи*, *шинные арбитры*, определяющие очередность доступа к системной магистрали, и другие устройства.

Логика работы системной магистрали, количество разрядов (линий) в шинах данных, адреса и управления, порядок разрешения конфликтных ситуаций, возникающих при одновременном обращении различных устройств ЭВМ к системной магистрали, образуют ***интерфейс системной шины***.

В *состав центральных устройств ЭВМ* входят: центральный процессор, основная память и ряд дополнительных узлов, выполняющих служебные функции: контроллер прерываний, таймер и контроллер прямого доступа к памяти (ПДП).

Периферийные устройства делятся на два вида: *внешние ЗУ* (НМД, НГМД, НМЛ) *и устройства ввода-вывода* (УВВ): клавиатура, дисплей, принтер, мышь, адаптер каналов связи (КС) и др.

Рассмотрим работу микропроцессора в одном цикле. Управляющая работой ЭВМ программа перед началом выполнения загружается в основную память (ОП). Адрес первой выполняемой команды передается микропроцессору и запоминается в счетчике команд.

Начало работы процессора заключается в том, что адрес из счетчика команд (в котором всегда хранится адрес очередной команды) выставляется на ША системной магистрали. Одновременно на ШУ выдается команда «**выборка из ОП»,** которая воспринимается основной памятью. Получив с ШУ системной магистрали команду, ОП считывает адрес с шины адреса, находит ячейку с этим номером и ее содержимое выставляет на ШД, а на ШУ выставляет сигнал о выполнении команды. Процессор, получив по ШУ сигнал об окончании работы ОП, вводит число с ШД на внутреннюю магистраль МП и через нее пересылает введенную информацию в регистр команд.

В регистре команд полученная команда разделяется на *кодовую* и *адресную части*. *Код команды* поступает в блок управления для выработки сигналов, настраивающих МП на выполнение заданной операции, и для определения адреса следующей команды (который сразу заносится в счетчик команд). *Адресная часть* команды выставляется на ША системной магистрали и сопровождается сигналом **«выборка из ОП»** на ШУ. Выбранная из ОП информация через шину данных поступает на внутреннюю магистраль МП, с которой вводится в арифметическое устройство (АУ). На этом заканчивается подготовка МП к выполнению операции, и начинается ее выполнение в АЛУ.

Результат выполнения операции выставляется микропроцессором на ШД, на ША выставляется адрес ОП, по которому этот результат необходимо записать, а на ШУ выставляется команда «**запись в ОП».** Получив с ШУ команду, ОП считывает адрес и данные с системной магистрали, организует запись данных по указанному адресу и после выполнения команды выставляет на ШУ сигнал, обозначающий, что число записано. Процессор, получив этот сигнал, начинает выборку очередной команды: выставляет адрес из счетчика команд на шину адреса, формирует команду **«выборка из ОП»** на ШУ и т.д.

В каждом цикле, получив команду в регистр команд и выделив код операции, процессор определяет, к какому устройству она относится. Если команда должна выполняться процессором, организуется ее выполнение по описанному циклу. Если же команда предназначена для выполнения в другом устройстве ЭВМ, центральный процессор (ЦП) передает ее соответствующему устройству. Процесс передачи команды другому устройству предусматривает следующие действия: ЦП выставляет на ША системной магистрали адрес интересующего его устройства. По ШУ передается сигнал «**поиск устройства».** Все устройства, подключенные к системной магистрали, получив этот сигнал, читают номер устройства с ША и сравнивают его со своим номером. Устройства, для которых эти номера не совпадают, на эту команду не реагируют. Устройство с совпавшим номером вырабатывает сигнал отклика по ШУ. ЦП, получив сигнал отклика, в простейшем случае выставляет имеющуюся у него команду на ШД и

сопровождает ее по шине управления сигналом «**передаю команду».** Получив сигнал о приеме команды, ЦП переходит к выполнению очередной своей команды, выставляя на ША содержимое счетчика команд.

В более сложных случаях, получив сигнал, что устройство откликнулось, прежде чем передавать команду, ЦП запрашивает устройство о его состоянии. Текущее состояние устройства закодировано в *байте состояния*, который откликнувшееся устройство передает процессору через ШД системной магистрали. Если устройство включено и готово к работе, то байт состояния – нулевой. Наличие в нем единиц свидетельствует о нештатной ситуации, которую ЦП пытается проанализировать и в необходимых случаях извещает оператора о сложившейся ситуации.

> Взаимодействие МП с внешними устройствами предусматривает выполнение логической последовательности действий, связанных с поиском устройства, определением его технического состояния, обменом командами и информацией. Эта логическая последовательность действий вместе с устройствами, реализующими ее, получила название ***интерфейс ввода-вывода***.

Для различных устройств могут использоваться разные логические последовательности действий, поэтому интерфейсов ввода-вывода может в одной и той же ЭВМ использоваться несколько. Если их удается свести к одному, универсальному, то такой интерфейс называется *стандартным.*

В IBM PC есть два стандартных интерфейса для связи ЦП с внешними устройствами:

- **параллельный** (типа Centronics)
- **последовательный** (типа RS-232).

Интерфейсы постоянно совершенствуются, поэтому с появлением новых ЭВМ, новых внешних устройств и даже нового программного обеспечения появляются и новые интерфейсы. Так, в программном обеспечении, разработанном ведущими фирмами, все шире используется новый интерфейс **«Plug and Play»** (Включи и играй), который предназначен для облегчения системной настройки ЭВМ при подключении новых устройств, к машине. Этот интерфейс позволяет подключить с помощью кабеля новое устройство, а после включения ЭВМ ее программное обеспечение автоматически определяет состав подключенных устройств, их типы в настраивает машину на работу с ними без вмешательства системного оператора.

Если при обращении ЦП к внешнему устройству продолжение выполнения основной программы центральным процессором возможно только после завершения операции ввода-вывода, то ЦП, запустив внешнее устройство, переходит в состояние ожидания и находится в нем до тех пор, пока внешнее устройство не сообщит ему об окончании обмена данными. Это приводит к простою большинства устройств ЭВМ, так как в каждый момент времени может работать только одно из них. Такой режим работы получил название *однопрограммного* - в каждый момент времени все устройства находятся в состоянии ожидания, и только одно устройство выполняет основную (и единственную) программу.

Для ликвидации таких простоев и повышения эффективности работы оборудования внешние устройства сделаны автономными. Получив от ЦП необходимую информацию, они самостоятельно организуют свою работу по обмену данными. Процессор же, запустив внешнее устройство, пытается продолжить выполнение программы. При необходимости (если встретятся соответствующие команды) он может запустить в работу несколько других устройств (так как внешние устройства работают значительно медленнее процессора). Если же ему приходится переходить в режим ожидания, то, пользуясь тем, что в ОП может одновременно находиться не одна, а несколько программ, ЦП переходит к выполнению очередной программы. При этом создается ситуация, когда в один и тот же момент времени различные устройства ЭВМ выполняют либо разные программы, либо разные части одной и той же программы, такой режим работы ЭВМ называется *многопрограммным.*

4.3 Организация работы ЭВМ при выполнении задания пользователя

Организация процессов ввода, преобразования и отображения результатов относится к сфере системного программного обеспечения. Это сложные процессы,, которые чаще всего делаются «прозрачными», т.е. незаметными для пользователя. Один из них – реализация задания пользователя: профессиональный пользователь (программист) пишет задание для ЭВМ в виде программы на *алгоритмическом языке.* Написанное задание (программа) представляет собой *исходный модуль,* сопровождаемый управляющими предложениями, указывающими операционной системе ЭВМ, на каком языке написана программа и что с ней надо делать. Если программа пишется на алгоритмическом языке, то управляющие предложения - на языке управления операционной системой.

- Исходный модуль перед исполнением должен быть переведен на внутренний язык машины. Эта операция выполняется специальной программой – *транслятором* (рисунок 4.1). Трансляторы выполняются в виде двух разновидностей:

интерпретаторы и *компиляторы*. Интерпретатор после перевода на язык машины каждого оператора алгоритмического языка немедленно исполняет полученную машинную программу.

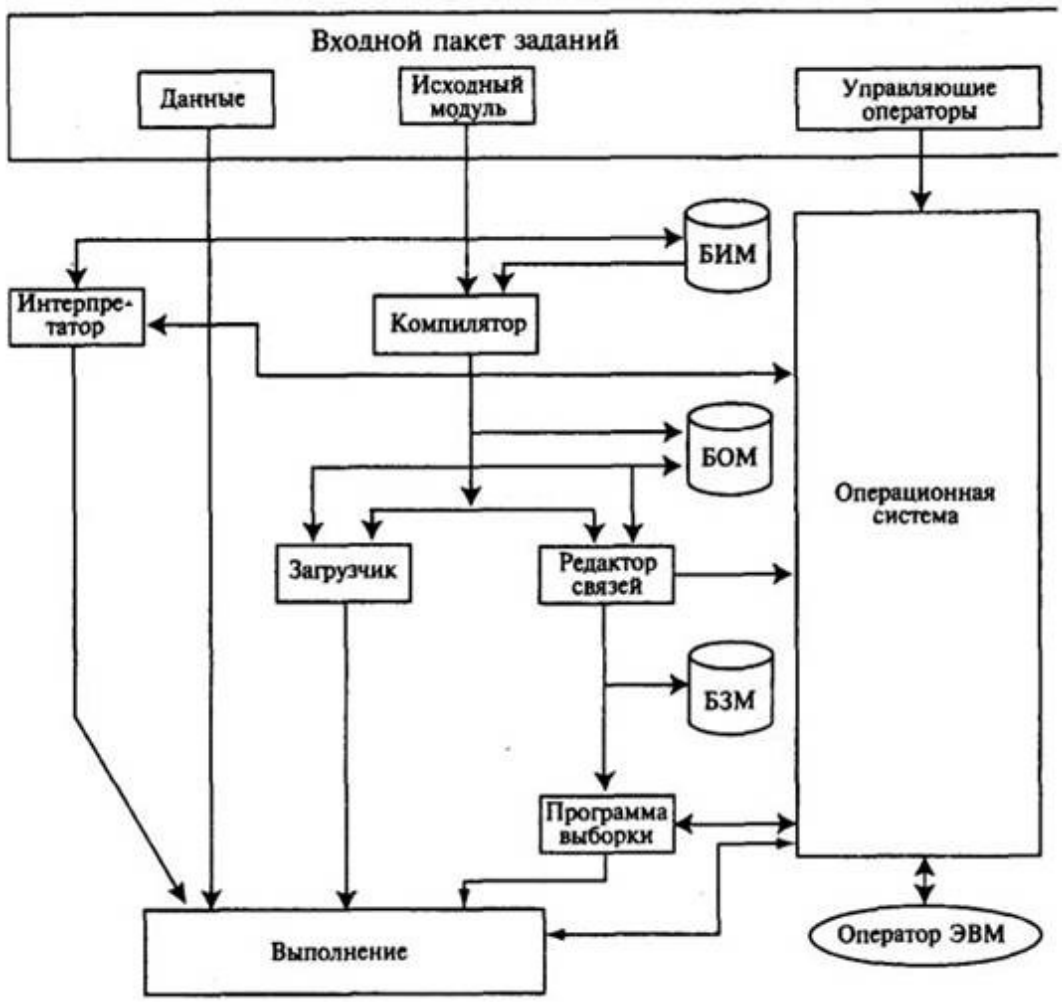


Рисунок 4.1 – Обработка заданий операционной системой

Компилятор же сначала полностью переводит всю программу, представленную ему в виде исходного модуля (ИМ), на язык машины. Получаемая при этом машинная программа представляет *собой объектный модуль* (ОМ). Результат работы компилятора может быть записан в библиотеку объектных модулей (БОМ) или передан другим программам для дальнейшей обработки, поскольку полученная машинная программа не готова к исполнению по двум причинам:

✦ Во-первых, она содержит *неразрешенные внешние ссылки* (т.е. обращение к программам, которые не содержатся в исходном модуле, но необходимы для работы основной программы, например, к стандартным программам алгоритмического языка, таким, как извлечение корня квадратного, вычисление тригонометрических функций и т.д.).

✦ Во-вторых, объектный модуль представляет собой машинную программу *в условных адресах* - каждый объектный модуль начинается с адреса Oh, тогда как для исполнения программа должна быть “привязана” к конкретным *физическим адресам* основной памяти.

Недостающие программы должны быть взяты из *библиотек компилятора* (которые могут быть написаны в виде исходных либо в виде объектных модулей) и добавлены к основной программе. Эту операцию выполняет *редактор связей.* В результате работы редактора связей образуется *загрузочный модуль* (ЗМ), который помещается в соответствующую библиотеку загрузочных модулей (БЗМ). В загрузочном модуле все ссылки разрешены, т.е. он содержит все необходимые стандартные программы, но привязки к памяти у загрузочного модуля нет.

Привязка к памяти загрузочного модуля производится *программой выборки,* которая переносит загрузочный модуль из библиотеки загрузочных модулей (обычно хранящейся на магнитном носителе) в основную память и во время этого переноса корректирует адреса, учитывая, с какого адреса основной памяти размещается загрузочный модуль. После перемещения загрузочного модуля в основную память программа выборки инициирует ее выполнение.

Представление машинной программы в виде исходных, объектных и загрузочных модулей позволяет реализовать наиболее эффективные программные комплексы. Например, если по одной и той же программе необходимо много раз производить расчеты, то неэффективно тратить каждый раз время на трансляцию и редактирование программы - ее нужно оформить в виде загрузочного модуля и хранить в соответствующей библиотеке. При обращении к такой программе сразу будет вызываться программа выборки для загрузки соответствующего модуля (а этапы компиляции и редактирования связей будут опускаться) - время на выполнение программы существенно сократится.

Если же программа только отлаживается или после каждого просчета ее нужно будет модернизировать, то получение загрузочного модуля и обращение к программе выборки будут лишними операциями. Для их обхода вместо редактора связей может быть применен *загрузчик* - программа, сочетающая в себе функции редактирования связей и загрузки полученной машинной программы в основную память для исполнения. Но при использовании загрузчика многократные просчеты по программе проводить невыгодно, так как каждый раз приходится выполнять лишние операции редактирования связей.

4.4.1 Отображение адресного пространства программы на основную память

Алгоритмы распределения, использования, освобождения ресурсов и предоставления к ним доступа предназначены для наиболее эффективной организации работы всего комплекса устройств ЭВМ. Рассмотрим их на примере управления основной памятью.

Для выполнения программы при ее загрузке в основную память ей выделяется часть *машинных ресурсов* - они необходимы для размещения команд, данных, управляющих таблиц и областей ввода-вывода, т.е. производится трансляция адресного пространства откомпилированной программы в местоположение в реальной памяти.

Выделение ресурсов может быть осуществлено самим программистом (особенно если он работает на языке, близком к машинному), но может производиться и операционной системой.

Если выделение ресурсов производится перед выполнением программы, такой процесс называется *статическим перемещением,* в результате которого программа "привязывается" к определенному месту в памяти вычислительной машины. Если же ресурсы выделяются в процессе выполнения программы, это называется *динамическим перемещением,* в этом случае программа не привязана к определенному месту в реальной памяти. Динамический режим можно реализовать только с помощью операционной системы.

При статическом перемещении может встретиться два случая:

1. Реальная память больше требуемого адресного пространства программы. В этом случае загрузка программы в реальную память производится, начиная с 0-го адреса (рисунок 4.2).

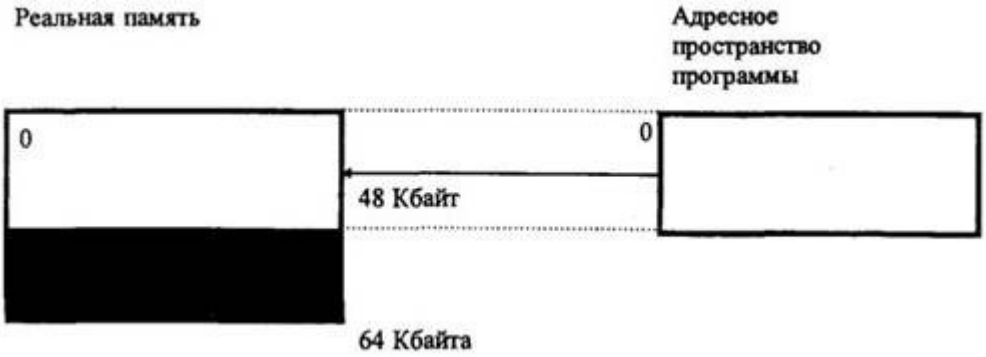


Рисунок 4.2 – Загрузка программы в реальную память (объем реальной памяти больше адресного пространства программы)

Загружаемая программа А является *абсолютной программой,* так как никакого изменения адресов в адресном пространстве, подготовленном компилятором, при загрузке в основную память не происходит - программа располагается с 0-го адреса реальной памяти.

2. Реальная память меньше требуемого адресного пространства программы (рисунок 4.3).

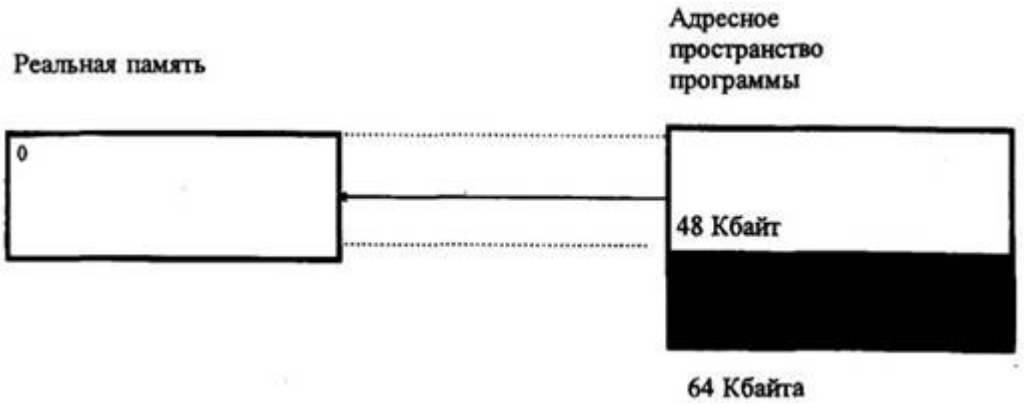


Рисунок 4.3 – Загрузка программы в реальную память
(объем реальной памяти меньше адресного пространства программы)

В этом случае программист (или операционная система) вынужден решать проблему, как организовать выполнение программы. Методов решения проблемы существует несколько: можно создать *оверлейную структуру* (т.е. разбить программу на части, вызываемые в ОП по мере необходимости), сделать модули программы *реентерабельными* (т.е. допускающими одновременную работу модуля по нескольким обращениям из разных частей программы или из различных программ) и т.д.

В некоторых операционных системах адреса откомпилированной (с 0 адреса) программы могут быть преобразованы в адреса реальной памяти, отличные от 0. При этом создается *абсолютный модуль,* который требует размещения его в памяти всегда с одного и того же адреса.

При мультипрограммном режиме, если имеем программы A, B и C, для которых известно, что программа A выполняется при размещении в памяти с адреса 60 Кбайт до 90 Кбайт, B - с 60 Кбайт до 90 Кбайт, C - с 50 Кбайт до 120 Кбайт, организовать их совместное выполнение невозможно, так как им необходим один и тот же участок реальной памяти. Эти программы будут ждать друг друга либо их нужно заново редактировать с другого адреса.

При работе в мультипрограммном режиме может сложиться ситуация, когда между программами образуются незанятые участки памяти. На рисунке 4.4 общий объем незанятой памяти, составляющий 50 Кбайт, достаточен, чтобы загрузить и программу D, находящуюся в ожидании. Но ее не удается загрузить, так как свободные участки памяти не являются смежными. Такое состояние называется *фрагментацией* реальной памяти. Оно характерно для систем со статическим перемещением.

Реальная память	
ОС	
20 Кбайт	
Программа А	0 Кбайт Программа D 50 Кбайт
10 Кбайг	
Программа В	
20 Кбайт	
Программа С	

Рисунок 4.4 – Фрагментация реальной памяти

В системах с динамическим перемещением программ перемещающий загрузчик размещает программу в свободной части памяти (рисунок 4.5) и допускает использование несмежных ее участков.

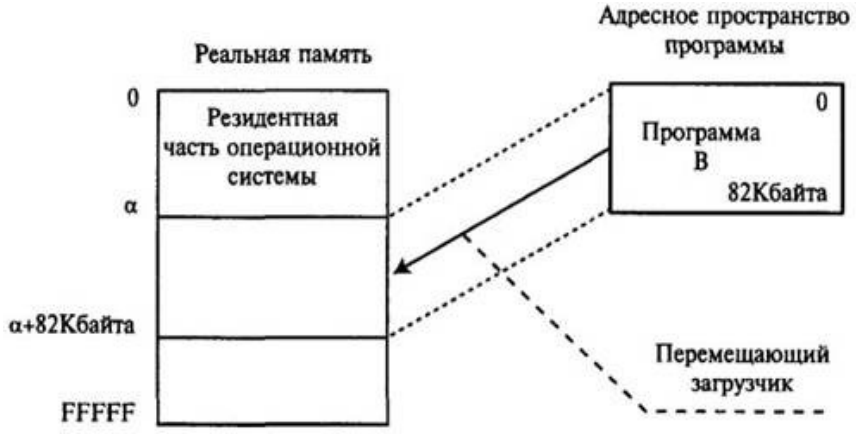


Рисунок 4.5 – Размещение программы в свободной части ОП

В этом случае имеется больше возможностей для организации мультипрограммной работы, а, следовательно, и для более эффективного использования временных ресурсов ЭВМ.

4.4.2 Адресная структура команд микропроцессора и планирование ресурсов

При больших размерах реализуемых программ возникают некоторые противоречия при организации мультипрограммного режима работы, трудности динамического распределения ресурсов.

В настоящее время разработано несколько способов решения этих противоречий. Например, для борьбы с фрагментацией основной памяти адресное пространство программы может быть разбито на отдельные *сегменты,* слабо связанные между собой. Тогда (рисунок 4.6) программа D общей длиной 50 Кбайт может быть представлена в виде

ряда сегментов, загружаемых в различные области ОП. Это позволяет использовать реальную память, теряемую из-за фрагментации.

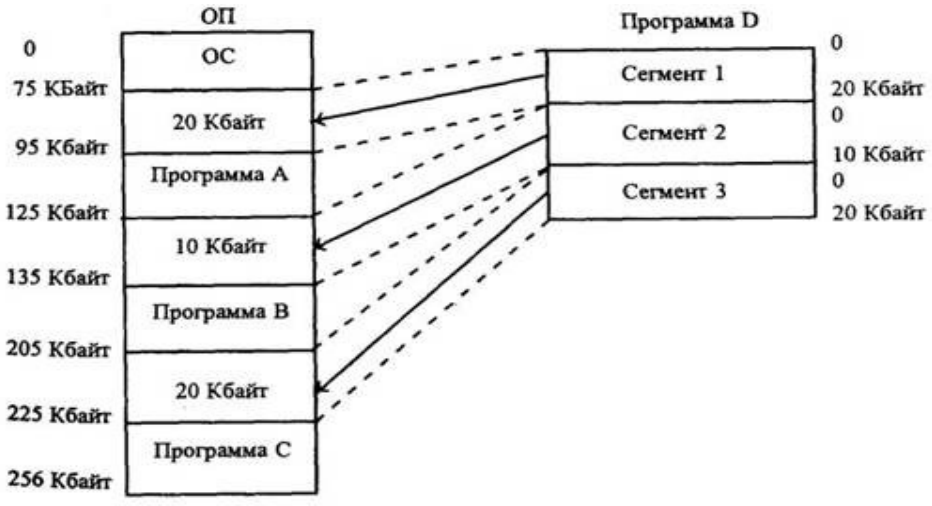


Рисунок 4.6 – Фрагментация ОП. Загрузка сегментированной программы

Адреса в каждом сегменте начинаются с 0. При статическом перемещении программы в процессе загрузки ее в основную память адреса должны быть привязаны к конкретному месту в памяти, на что уходит много времени, отвлекаются вычислительные ресурсы. Более эффективной является динамическая трансляция адресов (ДТА), которая заключается в том, что сегменты загружаются в основную память без трансляции адресного пространства (т.е. без изменения адресов в программе с учетом физического размещения в памяти команд и данных), а трансляция адресов каждой команды производится в процессе ее выполнения. Этот тип трансляции называется *динамическим перемещением* и осуществляется специальными аппаратурными средствами ДТА.

Каждый сегмент программы должен иметь свое имя. Форма имени сегмента может быть любой, например номер (рисунок 4.7 *а, б).*

При таком представлении адрес будет состоять из двух частей:

s - имя сегмента,

i - адрес внутри сегмента.

Если ЭВМ имеет 32-битовую адресную структуру, максимальная длина адреса в единственном сегменте будет длиной 32 разряда. Если 16 разрядов из 32 отвести под номер сегмента (а 16 - под смещение), то в этом случае все адресное пространство программы может состоять из 2^{16}= 64 Кбайта сегментов. Сегмент может содержать 2^{16}= 64 Кбайта (т.е. иметь адреса от 0 до 65535). При другой структуре адреса изменяются количество сегментов и их длина.

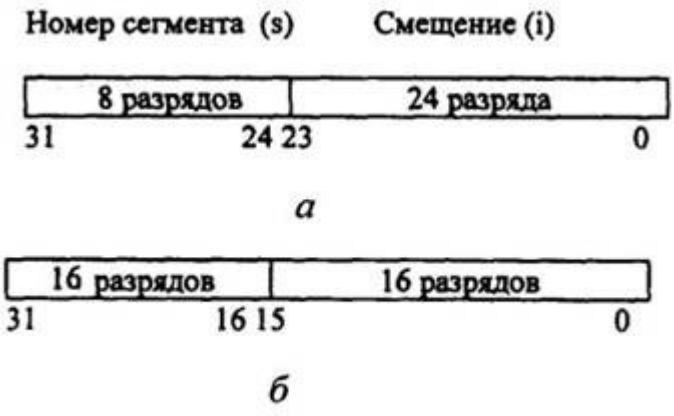


Рисунок 4.7 – Форма имени сегмента:

а - при выделении 8 разрядов; *б* - при выделении 16 разрядов

Структура адресов накладывает два важных **ограничения**:

- ограничивается максимальное число сегментов, которое может существовать в адресном пространстве программы;
- ограничивается максимальное смещение любого адреса в сегменте. При загрузке в основную память сегментированной программы каждый сегмент перемещается в реальную память отдельно, причем участки основной памяти могут быть или не быть смежными. Трансляция адресов не происходит - сегменты по-прежнему содержат свои относительные адреса.

Для динамической трансляции адресов (т.е. при определении абсолютных адресов по известным относительным, содержащим номер сегмента и смещение) операционная система строит специальные таблицы, устанавливающие соответствие между сегментируемым адресным пространством программы и действительными адресами сегментов в реальной памяти (рисунок 4.8).

Процессор может обращаться к основной памяти, используя только абсолютные адреса. Каждая строка таблицы сегментов содержит адрес начала сегмента в реальной памяти. Для каждого сегмента имеется одна строка таблицы. Таблицу сегментов содержит каждая выполняемая программа.

В дополнение к таблице сегментов для динамической трансляции адреса используется специальный управляющий регистр, называемый *регистром начала таблицы сегментов* (РНТС или STOR (segment table origin register)). В этот регистр занесен адрес таблицы сегментов выполняемой в данный момент программы.

На рисунке 4.9 изображено выполнение программы D. В РНТС находится адрес таблицы сегментов этой программы. Если программа B прервет выполнение программы D, то в РНТС будет занесен начальный адрес таблицы сегментов программы B.

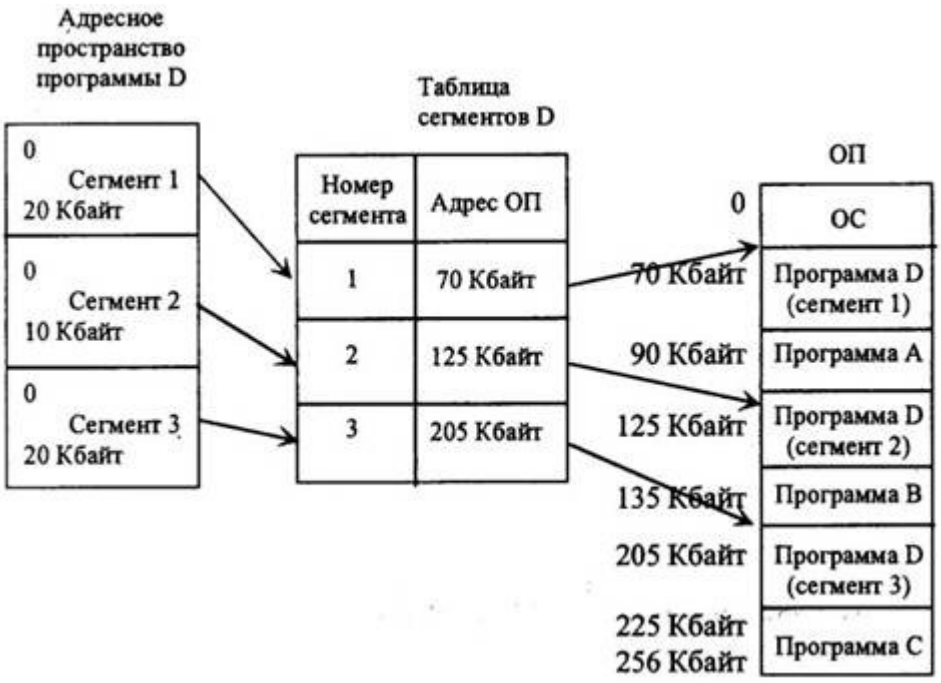


Рисунок 4.8 – Динамическая трансляция адресов при сегментной организации программы

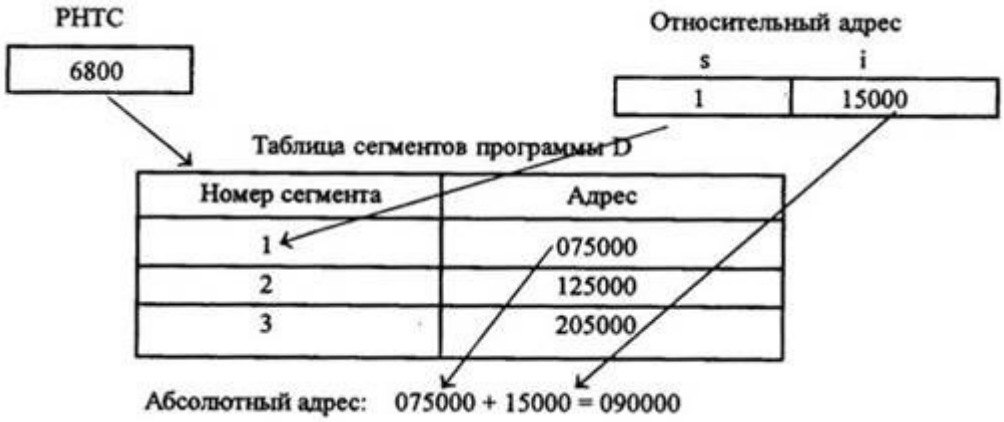


Рисунок 4.9 – Технология динамической трансляции адресов

Допустим, для выполняемой программы D начальный адрес таблицы сегментов 68000. В реальной вычислительной машине все действия выполняются в шестнадцатеричной системе счисления, мы же проведем вычисления для простоты в десятичной системе счисления.

Для обращения к адресу 15000 сегмента 1 производятся следующие действия:

- РНТС указывает на начало таблицы сегментов программы D - 68000;
- номер сегмента в относительном адресе используется как индекс при обращении к таблице сегментов. В данном примере обращение производится к 1-й строке;
- адрес, хранимый в выбранной строке таблицы сегментов, есть адрес начала сегмента в реальной памяти. Смещение в относительном адресе добавляется к

начальному адресу, и результат является адресом в реальной памяти: 15000+75000=90000. Для относительного адреса (сегмент 3, смещение 13000) будет получен абсолютный адрес 218000.

При ДТА такое определение адресов ведется в процессе выполнения каждой команды. Если операционной системе понадобится переместить исполняемую программу в другую часть памяти (например, чтобы исключить фрагментацию), сначала надо будет переслать команды и данные сегмента. Затем строку таблицы сегментов для данного сегмента нужно изменить так, чтобы она содержала новый адрес, и выполнение программы может быть продолжено. Это дает возможность динамического управления реальной памятью в процессе выполнения программы.

Использованием сегментации программ достигается уменьшение фрагментации основной памяти, но полностью фрагментация не устраняется – остаются фрагменты, длина которых меньше длины сегмента программы.

Если сегменты разделить на одну или несколько единиц, называемых *страницами,* которые имеют фиксированный размер, то поскольку размер страницы достаточно мал по сравнению с обычным размером сегментов, неиспользуемые фрагменты ОП значительно сокращаются в объеме - будет иметь место так называемая фрагментация внутри страниц. Следовательно, потери все-таки останутся, но они будут существенно меньше.

Сегментно-страничная организация добавляет еще один уровень в структуре адресного пространства программы. Теперь адресное пространство программы дробится на сегменты, внутри сегментов - на страницы и адреса внутри страниц. Структура адреса: (s, p, i) – рисунок 4.10, где s - имя сегмента адресного пространства программы; p - имя страницы; i - адрес внутри страницы.

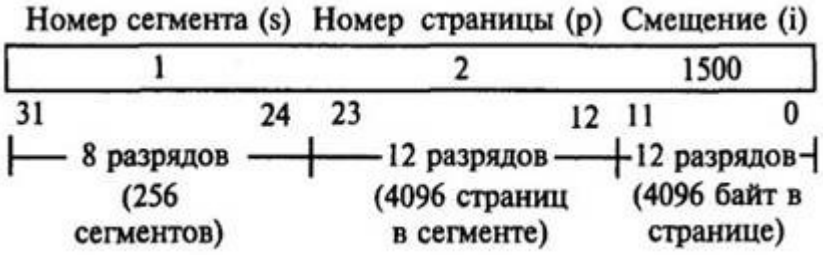


Рисунок 4.10 – Адресная структура при сегментно-страничной организации памяти внутри

Формирование сегментно-страничной структуры выполняется автоматически с помощью операционной системы. Для динамической трансляции адресов каждому сегменту необходимы одна таблица сегментов и несколько таблиц страниц (рисунок 4.12).

ДТА будет выполняться следующим образом:

- регистр начала таблицы сегментов содержит начальный адрес таблицы сегментов выполняемой программы 28000;
- номер сегмента в относительном адресе используется как индекс для обращения к записи таблиц сегментов. Эта запись идентифицирует начало таблицы страницы (реальный адрес) 30000;
- номер страницы в относительном адресе используется как индекс для обраще0ния к записи таблицы страниц. Эта запись идентифицирует начало страничного блока, содержащего эту страницу - 128000;
- смещение в относительном адресе и местоположение страничного блока объединяются вместе, формируя абсолютный адрес 129564. В реальной системе адрес страничного блока и смещение связываются, т.е. соединяются вместе для образования абсолютного адреса. Все преимущества динамического перемещения с использованием сегментации и страничной организации достигаются благодаря аппаратуре и программному обеспечению, а не пользователям системы. Специальные программы во время загрузки разбивают адресное пространство программы на сегменты и страницы, строят таблицы сегментов и страниц. Средства ДТА автоматически транслируют адрес в процессе выполнения программы.

4.4.3. Виртуальная память

Имея иерархическую структуру запоминающих устройств, на реальном объеме памяти, значительно меньшем максимального, можно имитировать работу с максимальной памятью. В этом случае программист работает так, как будто ему предоставлена реальная память максимально допустимого для данной ЭВМ объема, хотя имеющаяся реальная память значительно меньше по объему. Такой режим работы называется ***режимом виртуальной памяти***.

> Теоретически доступная пользователю ОП, объем которой определяется только разрядностью адресной части команды и которая не существует в действительности, называется ***виртуальной памятью***.

Виртуальная память имеет сегментно-страничную организацию и реализована в иерархической системе памяти ЭВМ. Часть ее размещается в *страничных блоках основной памяти,* а часть - в ячейках *внешней страничной памяти* (slot). Внешняя страничная память является частью внешней памяти.

> ***Ячейка (слот)*** - это записываемая область во внешней страничной памяти (например, на жестком магнитном диске). Она того же размера, что и страница.

Вычислительная система с 24-разрядным адресом может иметь адресное пространство в 16 777 2^{16} байт (16 Мбайт), с 32-разрядным адресом - 4 Гбайт. Структура такой памяти показана на рисунке 4.11.

Все программные страницы физически располагаются в ячейках внешней страничной памяти. Виртуальная же память существует только как продукт деятельности операционной системы (функционирующей на основе совместного использования внешней и страничной памяти).

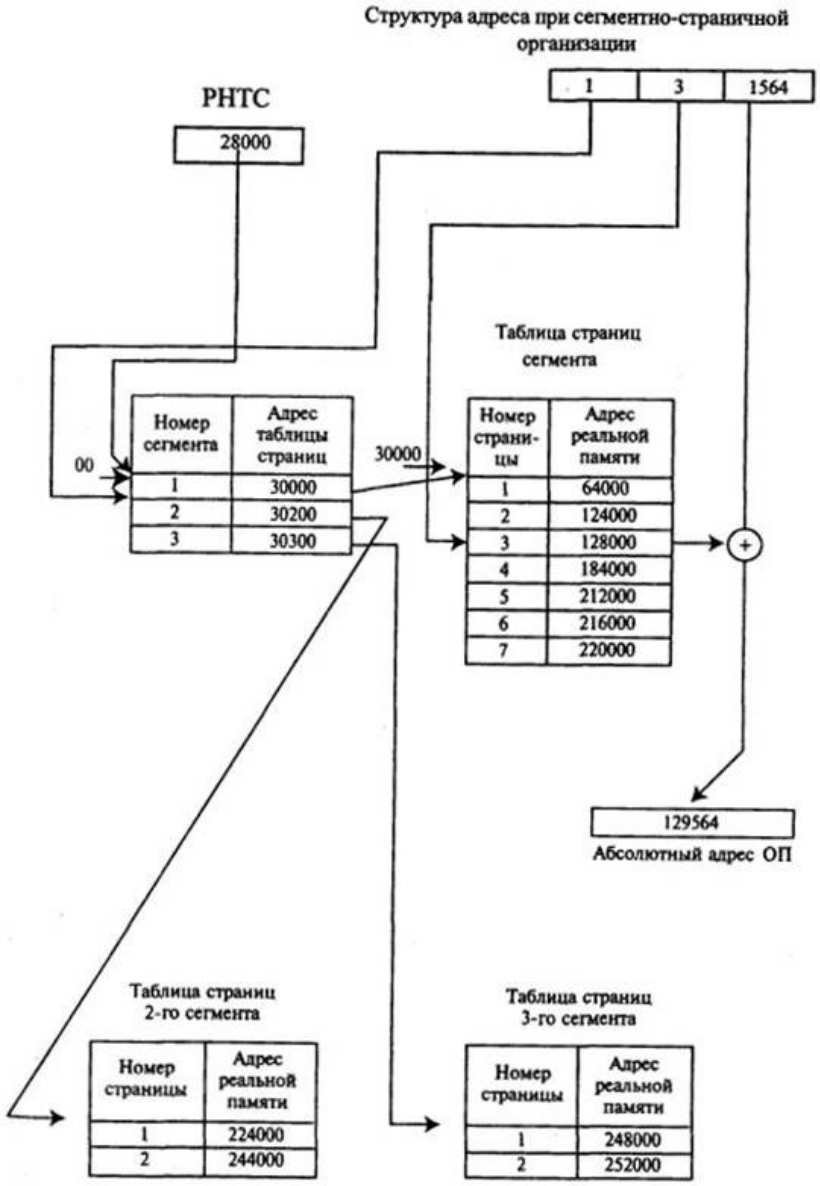


Рисунок 4.11 – Структурная схема формирования абсолютного адреса при сегментно-страничной организации ОП

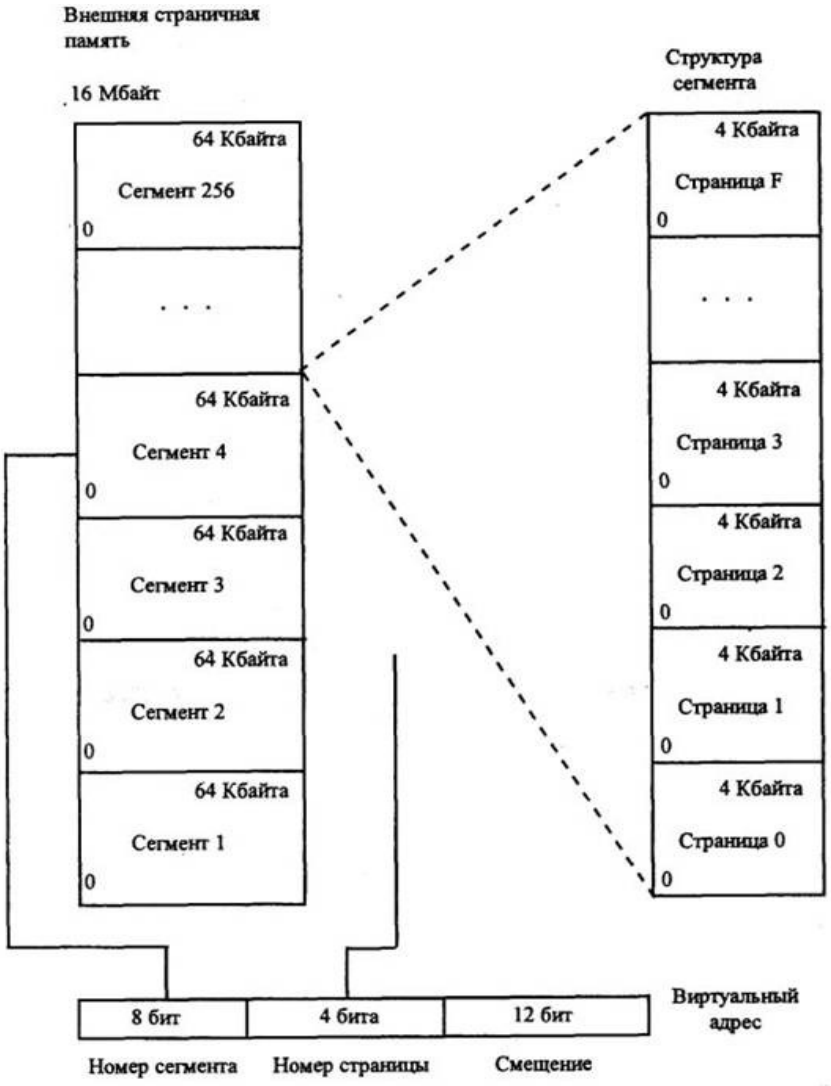


Рисунок 4.12 – Структура виртуальной памяти

Загрузить программу в виртуальную память - значит переписать несколько программных страниц из внешней страничной памяти в основную память. Если в процессе выполнения программы А система обнаружит, что требуемой страницы нет в реальной памяти, она должна переслать копию этой страницы из внешней страничной памяти в реальную память. Этот механизм называется *принудительным страничным обменом.*

При расшифровке виртуального адреса номер сегмента с помощью таблицы сегментов соотносится с адресом таблицы страниц. Таблица страниц содержит номер страницы и адрес страничного блока. В виртуальном режиме к таблице страниц добавляется еще одна колонка, содержащая бит недоступности. Нулевое состояние этого бита означает, что соответствующая страница загружена в реальную память. Единичное состояние означает, что страница недоступна, ее надо переписать в реальную память из

внешней. Местоположение страницы во внешней памяти указывается в таблице внешних страниц.

4.5 Система прерываний ЭВМ

Современная ЭВМ представляет собой комплекс автономных устройств, каждое из которых выполняет свои функции под управлением местного устройства управления независимо от других устройств машины. Включает устройство в работу центральный процессор. Он передает устройству команду и все необходимые для ее исполнения параметры. После начала работы устройства центральный процессор отключается от него и переходит к обслуживанию других устройств или к выполнению других функций.

Можно считать, что центральный процессор *переключает свое внимание* с устройства на устройство и с функции на функцию. На что именно обращено внимание ЦП в каждый данный момент, определяется выполняемой им программой.

Во время работы в ЦП поступает (и вырабатывается в нем самом) большое количество различных сигналов. Сигналы, которые выполняемая в ЦП программа способна воспринять, обработать и учесть, составляют *поле зрения* ЦП или другими словами - входят в *зону его внимания*.

Например, если процессором исполняется программа сложения двух двойных слов, которая анализирует регистр флагов ЦП, то в «поле ее зрения» находятся флаги микропроцессора, определяющие знаки исходных данных и результата, наличие переноса из тетрады или байта, переполнение разрядной сетки и др. Такая программа готова реагировать на любой из сигналов, находящихся в ее зоне внимания (а поскольку именно программа управляет работой ЦП, то она определяет и «зону внимания» центрального процессора). Но если во время выполнения такой программы нажать какую-либо клавишу, то эта программа «не заметит» сигнала от этой клавиши, так как он не входит в ее «поле зрения».

Для того чтобы ЦП, выполняя свою работу, имел возможность реагировать на события, происходящие вне его зоны внимания, наступления которых он «не ожидает», существует *система прерываний ЭВМ.* При отсутствии системы прерываний все заслуживающие внимания события должны находиться в поле зрения процессора, что сильно усложняет программы и требует большой их избыточности. Кроме того, поскольку момент наступления события заранее не известен, процессор в ожидании какого-либо события может находиться длительное время, и чтобы не пропустить его появления, ЦП не может «отвлекаться» на выполнение какой-либо другой работы. Такой режим работы

(режим сканирования ожидаемого события) связан с большими потерями времени ЦП на ожидание.

Кроме сокращения потерь на ожидание, режим прерываний позволяет организовать выполнение такой работы, которую без него реализовать просто невозможно. Например, при появлении неисправностей, нештатных ситуаций режим прерываний позволяет организовать работу по диагностике и автоматическому восстановлению в момент возникновения нештатной ситуации, прервав выполнение основной работы таким образом, чтобы сохранить полученные к этому времени правильные результаты. Тогда как без режима прерываний обратить внимание на наличие неисправности система могла только после окончания выполняемой работы (или ее этапа) и получения неправильного результата.

Таким образом, система прерываний позволяет микропроцессору выполнять основную работу, не отвлекаясь на проверку состояния сложных систем при отсутствии такой необходимости, или прервать выполняемую работу и переключиться на анализ возникшей ситуации сразу после ее появления.

Помимо требующих внимания нештатных ситуаций, которые могут возникнуть при работе микропроцессорной системы, процессору полезно уметь “переключать внимание” и на различные виды работ, одновременно выполняемые в системе. Поскольку управление работой системы осуществляется программой, этот вид прерываний должен формироваться программным путем.

В зависимости от места нахождения источника прерываний они могут быть разделены на *внутренние (программные и аппаратурные)* и *внешние* прерывания (поступающие в ЭВМ от внешних источников, например, от клавиатуры или модема).

Принцип действия системы прерываний заключается в следующем:

при выполнении программы после каждого рабочего такта микропроцессора изменяются содержимое регистров, счетчиков, состояние отдельных управляющих триггеров, т.е. изменяется состояние процессора.

информация о состоянии процессора лежит в основе многих процедур управления вычислительным процессом. Не вся информация одинаково актуальна, есть существенные элементы, без которых невозможно продолжение работы. Эта информация должна сохраняться при каждом «переключении внимания процессора.

Совокупность значений наиболее существенных информационных элементов называется *вектором состояния* или *словом состояния процессора* (в некоторых случаях она называется *словом состояния программы).*

Вектор состояния в каждый момент времени должен содержать информацию, достаточную для продолжения выполнения программы или повторного пуска ее с точки, соответствующей моменту формирования данного вектора.

Вектор состояния формируется в соответствующем регистре процессора или в группе регистров, которые могут использоваться и для других целей.

Наборы информационных элементов, образующих векторы состояния, отличаются у ЭВМ разных типов. В IBM PC вектор состояния включает содержимое счетчика команд, сегментных регистров, регистра флагов и аккумулятора (регистра AX).

При возникновении события, требующего немедленной реакции со стороны машины, ЦП прекращает обработку текущей программы и переходит к выполнению другой программы, специально предназначенной для данного события, по завершении которой возвращается к выполнению отложенной программы. Такой режим работы называется *прерыванием.*

Каждое событие, требующее прерывания, сопровождается специальным сигналом, который называется *запросом прерывания.* Программа, затребованная запросом прерывания, называется *обработчиком прерывания.*

Запросы на прерывание могут возникать из-за сбоев в аппаратуре (зафиксированных схемами контроля), переполнения разрядной сетки, деления на нуль, выхода за установленные для данной программы области памяти, затребования периферийным устройством операции ввода-вывода, завершения этой операции ввода-вывода или возникновения при этой операции особых условий и т.д.

Некоторые из этих запросов порождаются самой программой, но время их возникновения невозможно предсказать заранее.

При наличии нескольких источников запросов прерывания часть из них может поступать одновременно. Поэтому в ЭВМ устанавливается определенный порядок (дисциплина) обслуживания поступающих запросов. Кроме того, в ЭВМ предусматривается *возможность разрешать* или *запрещать* прерывания определенных видов.

ПЭВМ IBM PC может выполнять 256 различных прерываний, каждое из которых имеет свой номер (двухразрядное шестнадцатеричное число).

Все прерывания делятся на две группы: прерывания с номера 00h по номер IFh называются *прерываниями базовой системы ввода-вывода* (BIOS – Basic Input-Output System); прерывания с номера 20h по номер FFh называются *прерываниями DOS.* Прерывания DOS имеют более высокий уровень организации, чем прерывания BIOS, они строятся на использовании модулей BIOS в качестве элементов.

Прерывания делятся на три типа:

- аппаратные,
- логические
- программные.

Аппаратные прерывания вырабатываются устройствами, требующими внимания микропроцессора: прерывание № 2 - отказ питания; № 8 - от таймера; № 9 - от клавиатуры; № 12 - от адаптера связи; № 14 - от НГМД; № 15- от устройства печати и др.

Запросы на ***логические прерывания*** вырабатываются внутри микропроцессора при появлении “нештатных” ситуаций: прерывание № 0 - при попытке деления на 0; № 4 - при переполнении разрядной сетки арифметико-логического устройства; № 1 - при переводе микропроцессора в пошаговый режим работы; № 3 - при достижении программой одной из контрольных точек. Последние два прерывания используются отладчиками программ для организации пошагового режима выполнения программ (трассировки) и для остановки программы в заранее намеченных контрольных точках.

Запрос на ***программное прерывание*** формируется по команде **INT_n,** где n — номер вызываемого прерывания. Запрос на аппаратное или логическое прерывание вырабатывается в виде специального электрического сигнала.

ОРГАНИЗАЦИЯ ЭВМ И ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Методические указания к лабораторному практикуму «Основы языка Assembler»

Самара, 2013

СОДЕРЖАНИЕ

1 АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ КОМАНДЫ В АССЕМБЛЕРЕ

1.1 Арифметические команды

1.1.1 Сложение и вычитание

Команды ADD и SUB выполняют сложение и вычитание байтов или слов, содержащих двоичные данные. Вычитание выполняется в компьютере по методу сложения с двоичным дополнением: для второго операнда устанавливаются обратные значения бит и прибавляется 1, а затем происходит сложение с первым операндом. Во всем, кроме первого шага, операции сложения и вычитания идентичны.

Примеры показывают все пять возможных ситуаций:

- сложение/вычитание регистр-регистр;

add ax,bx

add ah,al

- сложение/вычитание память-регистр;

sub [000ah],ax

- сложение/вычитание регистр-память;

add ax, [000ah]

- сложение/вычитание регистр- непосредственное значение;

add ax,279

sub ch,3

- сложение/вычитание память - непосредственное значение.

sub cs:[000ah],3

> *Поскольку прямой операции память-память не существует, данная операция выполняется через регистр.*

1.1.2 Переполнения

Опасайтесь переполнений в арифметических операциях. Один байт содержит знаковый бит и семь бит данных, т. е. значения от -128 до +127. Результат

арифметической операции может легко превзойти емкость однобайтового регистра. Например, результат сложения в регистре AL, превышающий его емкость, автоматически не переходит в регистр AH. Предположим, что регистр AL содержит 60_{16}, тогда результат команды **Add AL, 20H** генерирует AL сумму - 80_{16}. Но операция также устанавливает флаг переполнения и знаковый флаг в состояние "отрицательно". Причина заключается в том, что 80_{16} или двоичное 1000 0000 является отрицательным числом. Т.е. в результате, вместо +128, мы получим -128. Так как регистр AL слишком мал для такой операции и следует воспользоваться регистром AX. Но полное слово имеет также ограничение: один знаковый бит и 15 бит данных, что соответствует значениям от -32768 до +32767.

1.1.3 Беззнаковые и знаковые данные

Для беззнаковых величин все биты являются битами данных, и вместо ограничения +32767 регистр может содержать числа до +65535. Для знаковых величин левый байт является знаковым битом. *Команды **ADD** и **SUB** не делают разницы между знаковыми и беззнаковыми величинами, они просто складывают и вычитают биты*. В следующем примере сложения двух двоичных чисел, первое число содержит единичный левый бит. Для беззнакового числа биты представляют положительное число 249, для знакового - отрицательное число -7:

Двоичное представление	Беззнаковое	Знаковое
11111001	249	-7
00000010	2	+2
11111011	251	-5

Двоичное представление результата сложения одинаково для беззнакового и знакового числа. Однако, биты представляют +251 для беззнакового числа и -5 для знакового. Таким образом, числовое содержимое поля может интерпретироваться по-разному.

Состояние "перенос" возникает в том случае, когда имеется перенос в знаковый разряд. Состояние "переполнение" возникает в том случае, когда перенос в знаковый разряд не создает переноса из разрядной сетки или перенос из разрядной сетки происходит без переноса в знаковый разряд. При возникновении переноса при сложении беззнаковых чисел, результат получается неправильный:

Двоичное представление	Беззнаковое	Знаковое	CF (перенос)	OF (переполнение)
A=11111100	252	-4		
B=00000101	5	+5		
A+B=00000001 неверно	1	1	1	0

При возникновении переполнения при сложении знаковых чисел, результат получается неправильный:

Двоичное представление	Беззнаковое	Знаковое	CF (перенос)	OF (переполнение)
A=01111001	121	+121		
B=00001011	11	+11		
A+B=10000100 неверно	132	-124	0	1

При операциях сложения и вычитания может одновременно возникнуть и переполнение, и перенос:

Двоичное представление	Беззнаковое	Знаковое	CF (перенос)	OF (переполнение)
A=11110110	246	-10		
B=10001001	137	-119		
A+B=01111111 неверно	127	+127	1	1

1.1.4 Умножение

Операция умножения для беззнаковых данных выполняется командой **MUL**, а для знаковых - **IMUL**. Ответственность за контроль над форматом обрабатываемых чисел и за выбор подходящей команды умножения лежит на самом программисте. Существуют две основные операции умножения:

«Байт на байт». Один из множителей находится в регистре AL, а другой в байте памяти или в однобайтовом регистре. После умножения произведение находится в регистре AX. Операция игнорирует и стирает любые данные, которые находились в регистре AH.

«Слово на слово». Один из множителей находится в регистре AX, а другой - в слове памяти или в регистре. После умножения произведение находится в двойном

слове, для которого требуется два регистра: старшая (левая) часть произведения находится в регистре DX, а младшая (правая) часть в регистре AX. Операция игнорирует и стирает любые данные, которые находились в регистре DX.

В единственном операнде команд MUL и IMUL указывается множитель. Рассмотрим следующую команду:

Mul multr

Если поле MULTR определено как байт (DB), то операция предполагает умножение содержимого AL на значение байта из поля MULTR. Если поле MULTR определено как слово (DW), то операция предполагает умножение содержимого AX на значение слова из поля MULTR. Если множитель находится в регистре, то длина регистра определяет тип операции, как это показано ниже:

Mul cl ;Байт-множитель: множимое в AL, произведение в AX

Mul bx ;Слово- множитель: множимое в AX, произведение в DX:AX

1.1.5 Беззнаковое умножение: Команда MUL

Команда MUL умножает беззнаковые числа.

Пример:

```
n db 10
. . .
mov al,2
mul n ;ax=2*10=20=0014h: ah=00h al=14h
mov al,26
mul n ;ax=26*10=260=0104h: ah=01h al=04h
```

1.1.6 Знаковое умножение: Команда IMUL

Команда IMUL умножает знаковые числа.

Пример:

```
mov ax,8
mov bx,-1
imul bx ; dx:ax=-8=0fffffff8h=0014h: dx=0ffffh ax=0fff8h
```

Таким образом, если множимое и множитель имеет одинаковый знаковый бит, то команды MUL и IMUL генерируют одинаковый результат. Но, если сомножители

имеют разные знаковые биты, то команда MUL вырабатывает положительный результат умножения, а команда IMUL - отрицательный.

Повышение эффективности умножения: При умножении на степень числа 2 (2,4,8 и т.д.) более эффективным является сдвиг влево на требуемое число битов. Сдвиг более чем на 1 требует загрузки величины сдвига в регистр CL. В следующих примерах предположим, что множимое находится в регистре AL или AX:

Умножение на 2: shl ax,1

Умножение на 8: mov ax,3

Shl ax,cl

1.1.7 Многословное умножение

Обычно умножение имеет два типа: "байт на байт" и "слово на слово". Как уже было показано, максимальное знаковое значение в слове ограничено величиной +32767. Умножение больших чисел требует выполнения некоторых дополнительных действий. Рассматриваемый подход предполагает умножение каждого слова отдельно и сложение полученных результатов.

Рассмотрим следующее умножение в десятичном формате:

```
      1  3  6  5
   x           1  2
   +  2  7  3  0
   1  3  6  5
-------------------
   1  8  3  8  0
```

Представим, что десятичная арифметика может умножать только двухзначные числа. Тогда можно умножить 13 и 65 на 12 раздельно, следующим образом:

```
      1  3          |        6  5
   x  1  2          |     x  1  2
   +  2  6          |     +1 3  0
   1  3             |     6  5
-------------------  |  -------------
   1  5  6          |     7  8  0
```

Следующим шагом сложим полученные произведения, но поскольку число 13 представляло сотни, то первое произведение в действительности будет 15600:

```
      1  5  6  0  0  |
   +        7  8  0  |
-------------------  |
   1  6  3  8  0  |
```

Ассемблерная программа использует аналогичную технику, за исключением того, что данные имеют размерность слов (четыре цифры) в шестнадцатеричном формате.

Умножение двойного слова на слово (z=x*y).

Пример:

```
X dd ?
Y dw ?
Z dw ? , ? , ?
. . .
wp equ word ptr
mov ax,wp x ;
mul y
mov z,ax
mov bx,dx
mov ax,wp x+2
mul y
add ax,bx
mov z+2,ax
adc dx,0
mov z+4,dx
```

1.1.8 Деление

Операция деления для беззнаковых данных выполняется командой **DIV**, а для знаковых - **IDIV**. Ответственность за подбор подходящей команды лежит на программисте. Существуют две основные операции деления:

Деление *«Слово на байт»*. Делимое находится в регистре AX, а делитель - в байте памяти или в однобайтовом регистре. После деления остаток получается в регистре AH, а частное - в AL. Так как однобайтовое частное очень мало (максимально+255 (FF_{16}) для беззнакового деления и +127 ($7F_{16}$) для знакового), то данная операция имеет ограниченное использование.

Деление *«Двойное слово на слово».* Делимое находится в регистровой паре DX:AX, а делитель - в слове памяти или в регистре. После деления остаток получается в регистре DX, а частное в регистре AX. Частное в одном слове допускает максимальное значение +32767 ($FFFF_{16}$) для беззнакового деления и +16383 ($7FFF_{16}$) для знакового.

В единственном операнде команд DIV и IDIV указывается делитель. Рассмотрим следующую команду:

div divisor

Если поле DIVISOR определено как байт (DB), то операция предполагает деление слова на байт. Если поле DIVISOR определено как слово (DW), то операция предполагает деление двойного слова на слово.

При делении, например, 13 на 3, получается результат 4 1/3. Частное есть 4, а остаток - 1. Заметим, что ручной калькулятор (или программа на языке BASIC) выдает в этом случае результат 4,333.... Значение содержит целую часть (4) и дробную часть (,333). Значение 1/3 и 333... есть дробные части, в то время как 1 есть остаток от деления.

1.1.9 Беззнаковое деление: Команда DIV

Команда DIV делит беззнаковые числа.

Пример

```
Mov ax,100
Mov bh,2
Div bh ; 100 div 2=50, ah=0 al=50
```

1.1.10 Переполнения и прерывания

Используя команды **DIV** и особенно **IDIV**, очень просто вызвать переполнение. Прерывания приводят (по крайней маре в системе, используемой при тестировании этих программ) к непредсказуемым результатам. В операциях деления предполагается, что частное значительно меньше, чем делимое. Деление на ноль всегда вызывает прерывание. Но деление на 1 генерирует частное, которое равно делимому, что может также легко вызвать прерывание.

Рекомендуется использовать следующее правило: если делитель - байт, то его значение должно быть меньше, чем левый байт (AH) делителя: если делитель - слово, то его значение должно быть меньше, чем левое слово (DX) делителя.

Проиллюстрируем данное правило для делителя, равного 1:

Операция деления: Делимое Делитель Частное

Слово на байт: 0123 01 (1)23

Двойное слово на слово: 0001 4026 0001 (1)4026

В обоих случаях частное превышает возможный размер. Для того чтобы избежать подобных ситуаций, полезно вставлять перед командами DIV и IDIV соответствующую проверку.

Для команды IDIV данная логика должна учитывать тот факт, что либо делимое, либо делитель могут быть отрицательными, а так как сравниваются абсолютные значения, то необходимо использовать команду NEG для временного перевода отрицательного значения в положительное.

1.1.11 Преобразование знака

Команда **NEG** обеспечивает преобразование знака двоичных чисел из положительного в отрицательное и наоборот.

Практически команда NEG устанавливает противоположные значения битов и прибавляет 1.

Примеры:

```
Neg ax
Neg bl
Neg BINAMT (байт или слово в памяти)
```

Преобразование знака для 35-битового (или большего) числа включает больше шагов. Предположим, что регистровая пара DX:AX содержит 32-битовое двоичное число. Так как команда NEG не может обрабатывать два регистра одновременно, то ее использование приведет к неправильному результату. В следующем примере показано использование команды NOT:

```
Not dx ;Инвертирование битов
Not ax ;Инвертирование битов
Add ax,1 ;Прибавление 1 к AX
Adc dx,0 ;Прибавление переноса к DX
```

1.2 Логические команды

1.2.1 Команды логических операций : and, not,or,xor,test

Логические операции являются важным элементом в проектировании микросхем и имеют много общего в логике программирования. Команды **AND**, **OR**, **XOR** и **TEST** являются командами логических операций. Эти команды используются для сброса и установки бит и для арифметических операций в коде ASCII . Все эти команды обрабатывают один байт или одно слово в регистре или в памяти, и устанавливают флаги ***CF, OF, PF, SF, ZF***.

AND: Если оба из сравниваемых битов равны 1, то результат равен 1; во всех остальных случаях результат - 0.

```
Mov al,00110011b
And al,11101110b ; al=0010010b
```

OR: Если хотя бы один из сравниваемых битов равен 1, то результат равен 1; если сравниваемые биты равны 0, то результат - 0.

```
mov al,00110011b
or al,11101110b ; al=11111111b
```

XOR: Если один из сравниваемых битов равен 0, а другой равен 1, то результат равен 1; если сравниваемые биты одинаковы(оба - 0 или оба - 1) то результат - 0.

```
mov al,00110011b
xor al,11101110b ; al=11011101b
```

TEST: действует как AND-устанавливает флаги, но не изменяет биты.

```
mov al,00110011b
test al,11101110b ; al=0011011b sf=0
```

Первый операнд в логических командах указывает на один байт или слово в регистре или в памяти и является единственным значением, которое может изменятся после выполнения команд.

1.2.2 Команды сдвига и циклического сдвига

Команды сдвига и циклического сдвига, которые представляют собой часть логических возможностей компьютера, имеют следующие свойства:

- обрабатывают байт или слово;
- имеют доступ к регистру или к памяти;
- сдвигают влево или вправо;
- сдвигают на величину до 8 бит (для байта) и 16 бит (для слова);
- сдвигают логически (без знака) или арифметически (со знаком).

Значение сдвига на 1 может быть закодировано как непосредственный операнд, значение больше 1 должно находиться в регистре CL.

Команды сдвига

При выполнении команд сдвига флаг CF всегда содержит значение последнего выдвинутого бита. Существуют следующие команды сдвига:

SHR	Логический (беззнаковый) сдвиг вправо
SHL	Логический (беззнаковый) сдвиг влево
SAR	Арифметический сдвиг вправо
SAL	Арифметический сдвиг влево

Следующий фрагмент иллюстрирует выполнение команды **SHR**:

```
Mov cl,03 ; AX:
Mov ax,10110111B ; 10110111
Shr ax,1 ; 01011011 ;Сдвиг вправо на 1
Shr ax,cl ; 00001011 ;Сдвиг вправо на 3
```

Первая команда SHR сдвигает содержимое регистра AX вправо на 1 бит. Выдвинутый в результате один бит попадает в флаг CF, а самый левый бит регистра AX заполняется нулем. Вторая команда сдвигает содержимое регистра AX еще на

три бита. При этом флаг CF последовательно принимает значения 1, 1, 0, а в три левых бита в регистре AX заносятся нули.

Рассмотрим действие команд арифметического вправо **SAR**:

```
Mov cl,03 ; AX:
Mov ax,10110111B ; 10110111
Sar ax,1 ; 11011011 ;Сдвиг вправо на 1
Sar ax,cl ; 11111011 ;Сдвиг вправо на 3
```

Команда SAR имеет важное отличие от команды SHR: для заполнения левого бита используется знаковый бит. Таким образом, положительные и отрицательные величины сохраняют свой знак.

В приведенном примере знаковый бит содержит единицу. При сдвигах влево правые биты заполняются нулями. Таким образом, результат команд сдвига SHL и SAL индентичен. Сдвиг влево часто используется для удваивания чисел, а сдвиг вправо - для деления на 2. Эти операции осуществляются значительно быстрее, чем команды умножения или деления.

Деление пополам нечетных чисел (например, 5 или 7) образует меньшие значения (2 или 3, соответственно) и устанавливают флаг CF в 1. Кроме того, если необходимо выполнить сдвиг на 2 бита, то использование двух команд сдвига более эффективно, чем использование одной команды с загрузкой регистра CL значением 2.

Команды циклического сдвига

Циклический сдвиг представляет собой операцию сдвига, при которой выдвинутый бит занимает освободившийся разряд.

Существуют следующие команды циклического сдвига:

ROR	Циклический (беззнаковый) сдвиг вправо
ROL	Циклический (беззнаковый) сдвиг влево
RCR	Циклический сдвиг вправо с переносом
RCL	Циклический сдвиг влево с переносом

Следующая последовательность команд иллюстрирует операцию циклического сдвига **ROR**:

```
Mov cl,03 ; BX:
```

```
Mov bx,10110111B ; 10110111
Ror bx,1 ; 11011011 ;Сдвиг вправо на 1
Rorbx,cl ; 01111011 ;Сдвиг вправо на 3
```

Первая команда ROR при выполнении циклического сдвига переносит правый единичный бит регистра BX в освободившуюся левую позицию. Вторая команда ROR переносит, таким образом, три правых бита.

В командах RCR и RCL в сдвиге участвует флаг CF. Выдвигаемый из регистра бит заносится в флаг CF, а значение CF при этом поступает в освободившуюся позицию.

Рассмотрим пример, в котором используются команды циклического и простого сдвига. Предположим, что 32-битовое значение находится в регистрах DX:AX так, что левые 16 бит лежат в регистре DX, а правые - в AX. Для умножения на 2 этого значения возможны следующие две команды:

```
Shl ax,1 ;Умножение пары регистров
Rcl dx,1 ; DX:AX на 2
```

Здесь команда SHL сдвигает все биты регистра AX влево, причем самый левый бит попадает в флаг CF. Затем команда RCL сдвигает все биты регистра DX влево и в освободившийся правый бит заносит значение из флага CF.

1.3 Примеры

Пример 1

Вычислить значение уравнения $y = \frac{12+3}{8+6} * 3 + 12$

Решение:

```
data segment
y db 0 ; описание переменной y в сегменте данных
data ends
st segment stack ‘stack’ ; описание сегмента стека
db 128 dup(?)
st ends
assume cs: code, ds: data, ss: st
code segment ;описание кодового сегмента
start:
mov ax,ds ; инициализация сегмента данных
mov ds,ax
mov ax,12 ; реализация сложения 12+3
add ax,3
mov bl,8 ; реализация сложения 8+6
add bl,6
div bl ;делим содержимое ax с содержимым bl
mov ah,0 ; остаток обнуляем и результат умножаем на 3
mov bl,3
mul bl
add ax,12 ; к произведению прибавляем 12 и заносим в y
mov y,al
mov ax,4c00h ;завершаем работу программы
int 21h
code ends
end start
```

Пример 2

Даны два числа в двоичном виде. Первое число проинвертировать и разделить на 4. второе число умножить на 2. Результаты логически сложить и первые четыре разряда заменить на противоположные.

Решение:

```
data segment
a db 10110101b
b db 00110111b
c db 0
data ends
st segment stack ‘stack’ ; описание сегмента стека
db 128 dup(?)
st ends
assume cs: code, ds: data, ss: st
code segment ;описание кодового сегмента
start:
mov ax,ds ; инициализация сегмента данных
mov ds,ax
not a ; инвертируем первое число и делим его на 4
shr a,2
shl b,1 ; второе число умножаем на 2
mov al , a ; полученные результаты складываем
add al , b
xor al, 00001111b ; меняем первые четыре разряда на противоположные
mov c,al
mov ax,4c00h ;завершаем работу программы
int 21h
code ends
end start
```

1.4 Варианты заданий

№ варианта	Задание 1 Вычислить значение уравнения	Задание 2
1.	$y=\frac{12/5+15/9}{8+6}+\frac{13/3-2/4}{3-2}$	Дано число в двоичном виде. Умножить его на 16. Результат перевернуть следующим образом: нулевой разряд становится седьмым, 1-ый становится 6-ым и т.д.
2.	$y=\frac{25}{13}-\frac{12/3-(12+4)/2}{5+2}$	Даны два числа в двоичном виде. В первом числе 3,5,7 разряды обнулить и результат разделить на 4, полученное значение логически умножить на 2-ое число.
3.	$y=\frac{16/3-8*2+3*5}{15/6}$	Дано число в двоичном виде. Поменять местами старшую и младшую части числа. Полученное значение разделить на 32 и проинвертировать.
4.	$y=\frac{5+9}{3}-\frac{16*3}{2+3}+\frac{23/3}{3}-2$	Даны два числа в двоичном виде. В первом числе старшие (4 разряда) разряды обнулить. Во втором числе сделать единицами 2,4,6 разряды. Полученные результаты логически перемножить.
5.	$y=\frac{33/13+12*2}{3*4}-\frac{4*6}{5}$	Дано число в двоичном виде. Разделить его на 16, занести в 1,3,7 разряды нули. Полученное значение логически сложить с числом 19.
6.	$y=\frac{11*2-6*4}{15-7}-\frac{29/4+7/4}{2+3}$	Даны два числа в двоичном виде. Первое число умножить на 9, второе разделить на 4. результаты логически перемножить и старшую часть поменять местами с младшей.
7.	$y=\frac{(22/3-13/5+3)/2}{5+6}-12$	Дано число в двоичном виде. Поменять местами третий бит с пятым. Результат умножить на 8 и проинвертировать.
8.	$y=\frac{(22*3-45/6)/14}{(134-7*5)/13}*2+4$	Дано число в двоичном виде. Логически перемножить его с числом 28. Проинвертировать результат и умножить на 4. В полученном значении 4,5,6 разряды заменить на противоположные.

9.	$y=\frac{(12-15/2)}{15/2}+12/7-\frac{13*2}{5}$	Даны два числа в двоичном виде. Логически их перемножить и в результирующем значении поменять местами 7-ой разряд с 1-ым, 5-ый со 2-ым.
10.	$y=\frac{5*3+13/4}{1+12/5}-28+\frac{13}{3}$	Дано число в двоичном виде. Поменять местами четные разряды с нечетным. Результат проинвертировать и умножить на 4.
11.	$y=\frac{12-9}{(16-9)/6}-\frac{3*2}{2+3}*(12/5)$	Дано число в двоичном виде. Вывернуть число «наизнанку» (разряды стоящие в середине сделать крайними). Результат разделить на 16.
12.	$y=\frac{14*3}{16/5-2*3}-\frac{14/4}{5-3}$	Даны два числа в двоичном виде. Из первого числа взять четыре младших разряда и поменять местами с четырьмя старшими разрядами второго числа. Результаты логически сложить и разделить на 8.
13.	$y=\frac{12/3+5*"+16/(8-5)}{15/2}$	Даны два числа в двоичном виде. Поменять местами 7,6,5,1- разряды первого числа с 0,2,3,4 разрядами второго числа соответственно. Результаты логически сложить и умножить на 8.
14.	$y=\frac{36+(14-5)*3}{(38-4*3)/3}*3-4$	Дано число в двоичном виде. Все нечетные разряды числа обнулить, а четные заменить на противоположные. Результат разделить на 4 и проинвертировать.
15.	$y=\frac{34+(15-7)*2}{36/4+35/8}-34*2$	Даны два числа в двоичном виде (первое число размером в байт , второе число размером в слово). Первое число умножить на 16 и в полученном значении обнулить 3,5 разряды. Результат сложить со старшей частью второго числа.
16.	$y=\frac{(123-4)/5+(6-3)/4}{36+4*3}-34$	Даны два числа в двоичном виде. Обнулить в первом числе 3,5,6 разряды и разделить полученное значение на 8, второе число умножить на 2 и логически сложить с первым. Результат проинвертировать.
17.	$y=\frac{34+(15-7)*2}{36/4+35/8}-34*2$	Дано двоичное число. В старшей части числа все четные биты заменить на противоположные. В младшей части числа все нечетные биты обнулить. Результат разделить на 16.
18.	$y=(24-14)*36+\frac{114-7*3}{136/5+7*3}$	Даны два числа в двоичном виде. В первом числе поменять местами старшую и младшую части числа. Во втором 1-ый и 4-ый разряды

		поменять местами с 3,7-мым разрядами соответственно. Результаты логически сложить и умножить на 4.
19.	$y = 37 - 14 * 3 * \frac{2*(26+3) - 44*6}{48 - 14/2}$	Дано число в двоичном виде. Разделить его на две составляющие: в первую войдут только четные разряды, во вторую только нечетные разряды. Их логически перемножить и результат умножить на 16.
20.	$y = \frac{(258-140)/(34-12)}{36+12/5} * (36+3) - 17/4$	Даны два числа в двоичном виде. Первое число умножить на 4. второе разделить на 2 . Результаты логически сложить. 0-ой и 7-ой разряды, в полученном значении, поменять местами
21.	$y = \frac{-15/6 + (34 - 7*2)*2}{16 + 13/8} - 14$	Дано число в двоичном виде. Поменять местами значения четных и нечетных разрядов. Полученное число проинвертировать и умножить на 8.
22.	$y = \frac{\frac{37+14*5}{2} + 5*6}{17 - 4/2} + 5*4$	Даны четыре числа в двоичном виде. Составить пятое число, которое состоит из 0-го и 1-го битов первого числа, 2-го и 3-го битов второго числа, 4,5-ые биты из третьего числа, 6,7-ой биты из четвертого числа. Полученное значение проинвертировать и разделить на 16.
23.	$y = 17*4 + \frac{5 - 8/3}{\frac{14+5}{4} + \frac{16*6+5}{10}}$	Дано двоичное число. Поменять местами 3-ий разряд с 7-ым. Полученное значение разделить на 8 и логически сложить с числом 56.
24.	$y = 35*4 + \frac{\frac{17-8/3}{2} + 4*8}{3+2}$	Дано число в двоичном виде. 0,1,4,5-ые разряды заменить на противоположные. Остальные занести в отдельный регистр, поставив их на 0,1,4,5-ые биты соответственно. Полученные значения логически перемножить
25.	$y = 3*6 + 4*4/3*(48 + 16/2) * \frac{5+8}{14-3}$	Даны два числа в двоичном виде. Поменять местами четные разряды одного числа с нечетными разрядами другого числа. Полученные значения логически перемножить и все разряды числа заменить на противоположные.

26.	$y=\frac{14-4}{2}*(5+16/4)*\frac{17+8/3}{\frac{8}{4}+2*5}$	Дано число в двоичном виде. Записать его наоборот. Полученное значение умножить на 8 и логически сложить с числом 17.
27.	$y=\frac{171+4*3*2}{\frac{36-4}{3}-17}+(5+6)*8*\frac{17}{5+2}$	Дано число в двоичном виде. Обнулить 2,3,4 разряды тремя способами.
28.	$y=\frac{250-4*6}{\frac{33+4}{2}-17}*(2+4)-17$	Дано число в двоичном виде. Заменить 0,1,3-ие разряды на противоположные. Все остальные разряды сделать единичными, результат разделить на 16.
29.	$y=7+4*6+5/2*13+\frac{\frac{14-4}{2}+17*6}{\frac{36-4}{3}+2}$	Даны два числа в двоичном виде. Поменять местами четные и нечетные разряды двумя способами.
30.	$y=\frac{3*4+5/7}{5-(3-4)/2}-\frac{17+2}{3*6+6}$	Дано число в двоичном виде. Все четные разряды сделать единицами. Полученные значения разделить на 8 и поменять местами правую и левую части.
31.	$y=\frac{25/4+5*6}{36}+\frac{37}{\frac{8*6+5}{3}+3*2}$	Даны два числа в двоичном виде. 1,3,7-ой разряды первого числа логически перемножить с 1,3,7-ым разрядом второго числа, результат разделить на 8 и проинвертировать.
32.	$y=\frac{37/2+\frac{4*6+5}{4}}{15-7}*(5+8)-136$	Даны два числа в двоичном виде. Все четные разряды первого числа логически сложить с четными разрядами второго числа, а нечетные обнулить. Результат разделить на 4 и проинвертировать.
33.	$y=\frac{3*4+5*6}{7/2+4/3+12/3}*\frac{16}{2}3*6$	Дано число в двоичном виде. К старшей части числа логически прибавить проинвертированную младшую части числа. Результат разделить на 2.
34.	$y=\frac{49/7+42/6+6*2}{8*5/9-4*6}+\frac{23+12}{26-4}*5$	Дано число в двоичном виде. К четным битам числа логически прибавить нечетные биты числа. Результат разделить на 8.

35.	$y=\dfrac{\dfrac{36+4*5}{2+6}-14/7+1}{\dfrac{36-4}{2}+12}-7*6*\dfrac{3+4}{12-7}$	Дано число в двоичном виде. Сделать 2,5,7-ой разряды единичными тремя способами.
36.	$y=\dfrac{\dfrac{2+3}{2}+\dfrac{4*6+25*5}{36-4}}{\dfrac{250*2-14}{37}}+(12-4)*(37-8)$	Даны два числа в двоичном виде. Поменять местами четные разряды первого числа с нечетными разрядами второго числа. Первое умножить на 2, а второе разделить на 2, результаты логически сложить.
37.	$y=\dfrac{\dfrac{320}{4}+\dfrac{256}{2}-14}{56*2}*(3+2)$	Даны два числа в двоичном виде. В первом числе 2,4,6,7-ой разряды заменить на противоположные. Второе число разделить на 8. результаты логически сложить и проинвертировать.
38.	$y=\dfrac{(12+4-7)*5}{36*4-\dfrac{17}{2}}*\dfrac{(2+5)}{7}+14$	Даны три числа в двоичном виде. Старшую часть первого числа логически сложить с младшей частью третьего числа, а младшую часть первого числа логически умножить на младшую часть второго числа. Результат разделить на 4.
39.	$y=\dfrac{34+260/2+170*3}{\dfrac{36+8}{2}+5*6}-\dfrac{34}{2}*(6+7)$	Даны два числа в двоичном виде. Получить третье число путем логического сложения четных разрядов первого числа и нечетных разрядов второго числа. Получить четвертое число путем логического умножения нечетных разрядов первого числа и четных разрядов второго числа. Третье и четвертое числа проинвертировать и разделить на 4.
40.	$y=\dfrac{15+30/2+3*5}{\dfrac{36+4}{2}+20-3}$	Даны два числа в двоичном виде. Первое число проинвертировать и разделить на 2. Второе число умножить на 4. Результаты логически сложить и первые четыре разряда заменить на противоположные.

2 АРИФМЕТИЧЕСКИЕ КОМАНДЫ И КОМАНДЫ ПЕРЕХОДОВ

2.1 Получение символов с клавиатуры

Ввод информации с клавиатуры - один из основных способов взаимодействия с компьютером IBM PC. DOS обеспечивает ряд функций, с помощью которых программа на ассемблере может обрабатывать нажатия клавиш.

Возможно, одним из наиболее простых способов получения символов клавиш является функция "Ввод с клавиатуры", то есть функция DOS номер 1. Функции DOS вызываются путем помещения номера функции в регистр AH и выполнения затем инструкции INT 21h. (Действительная работа инструкции INT несколько более сложна, но сейчас вам требуется только знать, что каждый раз при вызове функции DOS вы должны выполнять инструкцию INT 21h.) Следующий набранный на клавиатуре символ возвращается в регистре AL.

Например, когда выполняется код:

```
mov ah,1
int 21h
```

операционная система DOS помещает следующий набранный на клавиатуре символ в регистр AL. Заметим, что если клавиша не нажата, DOS будет ждать, когда она будет нажата; поэтому для выполнения данной функции может потребоваться неопределенное время.

2.2 Вывод символов на экран

Если нажатия клавиш означают взаимодействие пользователя с программным обеспечением, то экран является дополнением. IBM PC оснащаются дисплеями различных типов, начиная от цветного текстового до графического с высоким разрешением, но в данный момент мы рассмотрим только вывод символов.

Функция DOS с номером 2 обеспечивает наиболее непосредственный путь вывода символа на экран. Для этого нужно просто поместить 2 в регистр AH и выводимый символ в регистр DL, а затем вызвать DOS с помощью INT 21h. Следующий код отображает каждый введенный символ на экране:

```
mov ah,1
int 21h ; получить код следующей нажатой клавиши
mov ah,2
mov dl,al ; переместить считанный символ из AL в DL
int 21h ; вывести его на экран
```

Есть еще одно замечание, которое нужно сделать относительно клавиатуры, экрана и файлового ввода и вывода на языке Ассемблера. Те из вас, кто пользовался функциями scanf и printf в языке Си или функциями Readln n Writeln в Паскале, возможно с удивлением узнают, что в DOS не предусмотрено форматного ввода и вывода. DOS выполняет только посимвольный или построчный ввод-вывод. В Си для печати целой переменной вам требуется сделать следующее:

```
printf("\\d\n",i);
```

Си автоматически преобразует целое значение, которое хранится в 16-битовой ячейке памяти, в строку символов кода ASCII и печатает символы. В Ассемблере ваша программа должна явно преобразовывать переменные в строки символов, перед тем, как вывести их на экран. Аналогично, DOS знает только, как считывать символы и строки символов с клавиатуры, поэтому вам придется писать программы, преобразующие вводимые пользователем строки и символы в другие данные.

2.3 Безусловные переходы

Основной инструкцией перехода в наборе инструкций процессора 8086 является инструкция JMP. Эта инструкция указывает процессору 8086, что в качестве следующей за JMP инструкцией нужно выполнить инструкцию по целевой метке. Например, после завершения выполнения фрагмента программы:

```
mov ax,1
jmp AddTwoToAX
AddOneToAx:
 inc ax
 jmp AXIsSet
AddTwoToAX:
 inc ax
AXIsSet:
```

регистр AX будет содержать значение 3, а инструкции ADD и JMP, следующие за меткой AddOneToAX, никогда выполнены не будут. Здесь инструкция:

```
jmp AddTwoToAX
```

указывает процессору 8086, что нужно установить указатель инструкций IP в значение смещения метки AddTwoToAX; поэтому следующей выполняемой инструкцией будет инструкция:

```
add ax,2
```

Иногда совместно с инструкцией JMP используется операция SHORT. Для указания на целевую метку инструкция JMP обычно использует 16-битовое смещение. Операция SHORT указывает Турбо Ассемблеру, что нужно использовать не 16-битовое, а 8-битовое смещение (что позволяет сэкономить в инструкции JMP один байт). Например, последний фрагмент программы можно переписать так, что он станет на два байта короче:

```
mov ax,1
 jmp SHORT AddTwoToAX
AddOneToAx:
 inc ax
 jmp SHORT AXIsSet
AddTwoToAX:
 inc ax
AXIsSet:
```

Недостаток использования операции SHORT (короткий) состоит в том, что короткие переходы могут осуществлять передачу управления на метки, отстоящие от инструкции JMP не далее, чем на 128 байтов, поэтому в некоторых случаях Турбо Ассемблер может сообщать вам, что метка недостижима с помощью короткого перехода. К тому же операцию SHORT имеет смысл использовать для ссылок вперед, поскольку для переходов назад (на предшествующие метки) Турбо Ассемблер автоматически использует короткие переходы, если на метку можно перейти с помощью короткого перехода, и длинные в противном случае.

2.4 Условные переходы

Описанные в предыдущем разделе инструкции переходов - это только часть того, что вам потребуется для написания полезных программ. В действительности необходима возможность писать такие программы, которые могут принимать решения. Именно это можно делать с помощью операций условных переходов.

Инструкция условного перехода может осуществлять или нет переход на целевую (указанную в ней) метку, в зависимости от состояния регистра флагов. Рассмотрим следующий пример:

```
mov ah,1 ;функция DOS ввода с клавиатуры
 int 21h ; получить следующую нажатую клавишу
 cmp al,'A' ; была нажата буква "A"?
 je AWasTyped ; да, обработать ее
 mov [TampByte],al ; нет, сохранить символ
 .
 .
 .
AWasTyped:
 push ax ; сохранить символ в стеке
```

Сначала в данной программе с помощью функции операционной системы DOS воспринимается нажатая клавиша. Затем для сравнения введенного символа с символом A используется инструкция CMP. Эта инструкция аналогична инструкции SUB, только ее выполнение ни на что не влияет, поскольку назначение данной инструкции состоит в том, чтобы можно было сравнить два операнда, установив флаги так же, как это делается в инструкции SUB. Поэтому в предыдущем примере флаг нуля устанавливается в значение 1 только в том случае, если регистр AL содержит символ A.

Теперь мы подошли к основному моменту. Инструкция JE представляет инструкцию условного перехода, которая осуществляет передачу управления только в том случае, если флаг нуля равен 1. В противном случае выполняется инструкция, непосредственно следующая за инструкцией JE (в данном случае - инструкция MOV). Флаг нуля в данном примере будет установлен только в случае нажатия

клавиши A; и только в этом случае процессор 8086 перейдет к выполнению инструкции с меткой AWasTyped, то есть инструкции PUSH.

Набор инструкций процессора 8086 предусматривает большое разнообразие инструкций условных переходов, что позволяет вам осуществлять переход почти по любому флагу или их комбинации. Можно осуществлять условный переход по состоянию нуля, переноса, по знаку, четности или флагу переполнения и по комбинации флагов, показывающих результаты операций чисел со знаками.

Перечень инструкций условных переходов приводится в таблице 1.

Таблица 1 - Инструкции условных переходов

Название	**Значение**	**Проверяемые флаги**
JB/JNAE	Перейти, если меньше / перейти, если не больше или равно	CF = 1
JAE/JNB	Перейти, если больше или равно / перейти, если не меньше	CF = 0
JBE/JNA	Перейти, если меньше или равно / перейти, если не больше	CF = 1 или ZF = 1
JA/JNBE	Перейти, если больше / перейти, если не меньше или равно	CF = 0 и ZF = 0
JE/JZ	Перейти, если равно	ZF = 1
JNE/JNZ	Перейти, если не равно	ZF = 0
JL/JNGE	Перейти, если меньше чем / перейти, если не больше чем или равно	SF = OF
JGE/JNL	Перейти, если больше чем или равно /перейти, если не меньше чем	SF = OF
JLE/JNLE	Перейти, если меньше чем или равно / перейти, если не больше, чем	ZF = 1 или SF = OF
JG/JNLE	Перейти, если больше чем / перейти, если не меньше чем или равно	ZF = 0 или SF = OF
JP/JPE	Перейти по четности	PF = 1
JNP/JPO	Перейти по нечетности	PF = 0
JS	Перейти по знаку	SF = 1

JNS	Перейти, если знак не установлен	SF = 0
JC	Перейти при наличии переноса	CF = 1
JNC	Перейти при отсутствии переноса	CF = 0
JO	Перейти по переполнению	OF = 1
JNO	Перейти при отсутствии переполнения	OF = 0

CF - флаг переноса, SF - флаг знака, OF - флаг переполнения, ZF - флаг нуля, PF - флаг четности.

Не смотря на свою гибкость, инструкции условного перехода имеют также серьезные ограничения, поскольку переходы в них всегда короткие. Другими словами, целевая метка, указанная в инструкции условного перехода, должна отстоять от инструкции перехода не более, чем на 128 байт. Например, Турбо Ассемблер не может ассемблировать:

```
JumpTarget:
.
.
.
DB 1000 DUP (?)
.
.
.
dec ax
jnz JumpTarget
```

так как метка JumpTarget отстоит от инструкции JNZ более чем на 1000 байт. В данном случае нужно сделать следующее:

```
JumpTarget:
.
.
.
DB 1000 DUP (?)
.
.
.
dec ax
jnz SkipJump
jmp JumpTarget
SkipJump:
```

где условный переход применяется для того, чтобы определить, нужно ли выполнить длинный безусловные переход.

Командам условного перехода может предшествовать любая команда, изменяющая состояние флагов.

2.5 Пример

Рассмотрим простейший пример с использованием ввода, вывода и различных видов переходов.

Необходимо ввести с клавиатуры значение двух переменных a и x. Если a<x, то сложить их значения, а иначе из x отнять a.

```
data segment
a db ?
x db ?
per db 10,13,'$'
mesa db 10,13,'Input a: $'
mesx db 10,13,'Input x: $',10,13
data ends
s segment stack
db 128 dup(?)
s ends
code segment
main:
assume ss:s,ds:data,cs:code
mov ax,data
mov ds,ax
mov dx,offset mesa
mov ah,9 ;Приглашение на ввод a
int 21h

mov ah,1 ;Считывание нажатого символа
int 21h

mov a,al ;Запись считанного символа в a

mov dx,offset mesx
mov ah,9 ;Приглашение на ввод x
int 21h

mov ah,1 ;Считывание нажатого символа
int 21h
mov x,al ;Запись считанного символа в x

mov dx,offset per
```

```
mov ah,9 ;Перевод строки
int 21h

mov al,x
cmp a,al
jl Lower ;Если a<x,то перейти на метку Lower. Иначе на метку Higher
Higher:
mov al,a
sub al,x
add al,30h ;Коррекция по вычитанию
jmp short l1
lower:
mov al,x ;В регистр al записываем результат сложения a и x
add al,a
sub al,30h ;Корекция по сложению
l1:
mov dl,al
mov ah,2 ;Вывод содержимого dl на экран
int 21h

mov ah,0 ;Ожидание нажатия клавиши
int 16h

mov ah,4ch
int 21h
code ends
end main
```

Остановимся подробнее на строках, в которых происходит коррекция по сложению и вычитанию. Так как Ассемблер не способен обрабатывать просто десятичные числа, то необходимо придумывать алгоритмы обработки самостоятельно. Удобнее всего приводить их к нераспакованному десятичному виду.

При чтении с клавиатуры происходит чтение именно символа, и в регистр записывается его код. Например у чисел 1 и 5 коды соответственно 31h и 35h. Чтобы привести к нераспакованному десятичному виду необходимо привести их к виду 01h и 05h. Эти коды имеют символы отличные от 1 и 5, но над ними гораздо удобнее выполнять арифметические операции. Существуют специальные команды коррекции, но о них вы узнаете позже.

2.6 Задания

В соответствии с вариантом напишите программу на языке ассемблера с полным описанием сегментов для вычисления значения y. Используйте, где требуется, 32х-разрядные регистры.

1	$y = y1 + y2;\ y1 = \begin{cases} a + x, & \text{если } x > a \\ 2a - x, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a * x, & \text{если } x > 10 \\ x, & \text{если } x <= 10 \end{cases}.$
2	$y = y1 - y2;\ y1 = \begin{cases} x - 2, & \text{если } x >= 2 \\ 8, & \text{если } x < 2 \end{cases};\ y2 = \begin{cases} 4, & \text{если } x = 0 \\ a - x, & \text{если } x <> 0 \end{cases}.$
3	$y = y1 * y2;\ y1 = \begin{cases} x - a, & \text{если } x > a \\ 5, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a, & \text{если a} > x \\ a * x, & \text{если a} <= x \end{cases}.$
4	$y = y1 + y2;\ y1 = \begin{cases} 2 - x, & \text{если } x < 2 \\ a + 3, & \text{если } x >= 2 \end{cases};\ y2 = \begin{cases} a - 1, & \text{если } x < a \\ a * x - 1, & \text{если } x >= a \end{cases}.$
5	$y = y1 - y2;\ y1 = \begin{cases} \lvert x\rvert, & \text{если } x < 0 \\ x - a, & \text{если } x >= 0 \end{cases};\ y2 = \begin{cases} a + x, & \text{если } x \bmod 3 = 1 \\ 7, & \text{в остальных случаях} \end{cases}.$
6	$y = y1 + y2;\ y1 = \begin{cases} x \bmod 4, & \text{если } x > a \\ a, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a * x, & \text{если } x / a > 3 \\ x, & \text{если } x / a <= 3 \end{cases}.$
7	$y = y1 + y2;\ y1 = \begin{cases} 4 - x, & \text{если } \lvert x\rvert < 3 \\ a + x, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} 2, & \text{если } x \text{ четное} \\ a + 2, & \text{в остальных случаях} \end{cases}.$
8	$y = y1 + y2;\ y1 = \begin{cases} 4 * x, & \text{если } x <= 4 \\ x - a, & \text{если } x > 4 \end{cases};\ y2 = \begin{cases} 7, & \text{если } x \text{ нечетное} \\ x / 2 + a, & \text{в остальных случаях} \end{cases}.$
9	$y = y1 * y2;\ y1 = \begin{cases} a * x, & \text{если } x \bmod 3 = 2 \\ 9, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} a - x, & \text{если } a > x \\ a + 2, & \text{если } a <= x \end{cases}.$
10	$y = y1 - y2;\ y1 = \begin{cases} a + \lvert x\rvert, & \text{если } x > a \\ a - 7, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a * 3, & \text{если } a > 3 \\ 11, & \text{если a} <= 3 \end{cases}.$
11	$y = y1 \bmod y2;\ y1 = \begin{cases} 10 + x, & \text{если } x > 1 \\ \lvert x\rvert + a, & \text{если } x <= 1 \end{cases};\ y2 = \begin{cases} 2, & \text{если } x > 4 \\ x, & \text{если } x <= 4 \end{cases}.$
12	$y = y1 / y2;\ y1 = \begin{cases} 15 + x, & \text{если } x > 7 \\ \lvert a\rvert + 9, & \text{если } x <= -7 \end{cases};\ y2 = \begin{cases} 3, & \text{если x} > 2 \\ \lvert x\rvert - 5, & \text{если } x <= 2 \end{cases}.$
13	$y = y1 * y2;\ y1 = \begin{cases} 3 + x, & \text{если } x = a \\ a - x, & \text{если } x <> a \end{cases};\ y2 = \begin{cases} \lvert a\rvert, & \text{если } a < x \\ \lvert a\rvert - x, & \text{если ```a} >= x \end{cases}.$
14	$y = y1 - y2;\ y1 = \begin{cases} 2 * x + a, & \text{если } x > 2 \\ 2 * x + 1, & \text{если } x <= 2 \end{cases};\ y2 = \begin{cases} \lvert x\rvert + 1, & \text{если x} > 0 \\ a - 1, & \text{если } x <= 0 \end{cases}.$
15	$y = y1 \bmod y2;\ y1 = \begin{cases} 8 + \lvert x\rvert, & \text{если } x < 1 \\ \lvert a\rvert * 2, & \text{если } x >= 1 \end{cases};\ y2 = \begin{cases} 3, & \text{если } x = a \\ a + 1, & \text{если } x <> a \end{cases}.$
16	$y = y1 + y2;\ y1 = \begin{cases} 4 + x, & \text{если } x <= 3 \\ a * x, & \text{если } x > 3 \end{cases};\ y2 = \begin{cases} \lvert a\rvert - 2, & \text{если x} > a \\ x, & \text{если } x <= a \end{cases}.$
17	$y = y1 - y2;\ y1 = \begin{cases} a + \lvert x\rvert, & \text{если } x < 0 \\ x - a, & \text{если } x >= 0 \end{cases};\ y2 = \begin{cases} 7, & \text{если x} < 3 \\ a, & \text{если } x >= 3 \end{cases}.$
18	$y = y1 \bmod y2;\ y1 = \begin{cases} 7 + x, & \text{если } x < 3 \\ \lvert a\rvert + x, & \text{если } x >= 3 \end{cases};\ y2 = \begin{cases} 1, & \text{если } x > 5 \\ a + x, & \text{если } x <= 5 \end{cases}.$
19	$y = \lvert y1\rvert + y2;\ y1 = \begin{cases} -5, & \text{если } x > 4 \\ x - a, & \text{если } x <= 4 \end{cases};\ y2 = \begin{cases} \lvert a\rvert, & \text{если x} > a \\ 9, & \text{если } x <= a \end{cases}.$

20	$y = y1 * y2;\ y1 = \begin{cases} 2 * x, & \text{если } x < 5 \\ \lvert a \rvert + x, & \text{если } x >= 5 \end{cases};$
21	$y = y1 + \lvert y2 \rvert;\ y1 = \begin{cases} 3, & \text{если } x \bmod 3 = 1 \\ x - a, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} a / x, & \text{если } x <> 0 \\ 4, & \text{если } x = 0 \end{cases}.$
22	$y = y1 - y2;\ y1 = \begin{cases} \lvert x \rvert + \lvert a \rvert, & \text{если } x <= 3 \\ x * a, & \text{если } x > 3 \end{cases};\ y2 = \begin{cases} 3, & \text{если } a = x \\ a - x, & \text{если } a <> x \end{cases}.$
23	$y = y1 + y2;\ y1 = \begin{cases} 2 * x, & \text{если } \lvert x \rvert > 4 \\ 4 + a, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} 9, & \text{если } x = 0 \\ a / x, & \text{если } x <> 0 \end{cases}.$
24	$y = y1 * y2;\ y1 = \begin{cases} x, & \text{если } x \bmod 4 <> 2 \\ a + x, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} a - x, & \text{если } x < a \\ a * x, & \text{если } x >= a \end{cases}.$
25	$y = y1 / y2;\ y1 = \begin{cases} 12, & \text{если } x < 12 \\ x + 1, & \text{если } x >= 12 \end{cases};\ y2 = \begin{cases} 2, & \text{если } x > 2 \\ a + x, & \text{если } x <= 2 \end{cases}.$

3 ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕР ЗАДАЧ С ИСПОЛЬЗОВАНИЕМ МАССИВОВ СТРОКОВЫХ ДАННЫХ

3.1 Предопределенный идентификатор $

В Турбо Ассемблере имеется несколько предопределенных идентификаторов (например, @data). Еще один простой, но удивительно полезный предопределенный идентификатор - это идентификатор $, который всегда установлен в текущее значение счетчика адреса. Другими словами, идентификатор $ всегда равен текущему смещению в сегменте, в котором Турбо Ассемблер в данный момент выполняет ассемблирование. $ представляет собой постоянное значение смещения, аналогичное OFFSET MemVar. Это позволяет использовать $ в выражениях или в любом месте, где допускается использование константы.

Идентификатор $ очень удобно использовать для вычисления длины данных и кода. Предположим, например, что вы хотите приравнять идентификатор STRING_LENGTH к длине строки в байтах. Без предопределенного идентификатора $ вам придется сделать следующее:

```
StringStart LABEL BYTE
 db 0dh,0ah,'Текстовая строка'odh,0ah
 StringEnd LABEL BYTE
 STRING_LENGTH EQU (StringEnd-StringStart)
```

а с помощью идентификатора $ вы можете записать:

```
StringStart LABEL BYTE
 db 0dh,0ah,'Текстовая строка'odh,0ah
 STRING_LENGTH EQU ($-StringStart)
```

Длину (в словах) массива слов можно вычислить следующим образом:

```
WordArray DW 90h, 25h, 0, 16h, 23h
 WORD_ARRAY_LENGTH EQU (($-WordArray)/2)
```

Вы, конечно, можете сосчитать отдельные элементы вручную, но для больших массивов и строк это довольно затруднительно.

3.2 Цепочные команды

3.2.1 Инструкция LODS

Инструкция LODS, которая загружает байт или слово из памяти в аккумулятор (накопитель), подразделяется на две инструкции - LODSB и LODSW. Инструкция LODSB загружает байт, адресуемый с помощью пары регистров DS:SI, в регистр AL и уменьшает или увеличивает регистр SI (в зависимости от состояния флага направления). Если флаг направления равен 0 (установлен с помощью инструкции CLD), то регистр SI увеличивается, а если флаг направления равен 1 (установлен с помощью инструкции STD), то регистр SI уменьшается. И это верно не только для инструкции LODSB; флаг направления управляет направлением, в котором изменяются все регистры-указатели строковых инструкций.

Например, в следующем фрагменте программы:

```
cld
mov si,0
lodsb
```

инструкция LODSB загружает регистр AL содержимым байта со смещением 0 в сегменте данных и увеличивает значение регистра SI на 1.Это эквивалентно выполнению следующих инструкций:

```
mov si,0
mov al,[si]
inc si
```

Однако инструкция LODSB работает существенно быстрее (и занимает на два байта меньше), чем инструкции:

```
mov al,[si]
inc si
```

Инструкция LODSW аналогична инструкции LODSB. Она сохраняет в регистре AX слово, адресуемое парой регистров DS:SI; а значение регистра SI уменьшается или увеличивается на 2, а не на 1. Например, инструкции:

```
std
mov si,0
lodsw
```

загружают слово со смещением 10 в сегменте данных в регистр RU, а затем значение SI уменьшается на 2.

3.2.2 Инструкция STOS

Инструкция STOS - это дополнение инструкции LODS. Она записывает значение размером в байт или слово из аккумулятора в ячейку памяти, на которую указывает пара регистров ES:DI, а затем увеличивает или уменьшает DI. Инструкция STOSB записывает байт, содержащийся в регистре AL, в ячейку памяти по адресу ES:DI, а затем увеличивает или уменьшает регистр DI, в зависимости от флага направления. Например, инструкции:

```
std
mov di,0ffffh
mov al,55h
stosb
```

записывают значение 55h в байт со смещением 0FFFFh в сегменте, на который указывает регистр ES, а затем уменьшает DI до значения 0FFFEh.

Инструкция STOSW работает аналогично, записывая значение размером в слово, содержащееся в регистре AX, по адресу ES:DI, а затем увеличивает или уменьшает значение регистра DI на 2. Например, инструкции:

```
cld
mov di,0ffeh
mov al,102h
stosw
```

записывают значение 102h размером в слово, записанное в регистре AX, по смещению 0FFEh в сегменте, на который указывает регистр ES, а затем значение регистра DI увеличивается до 1000h.

Инструкции LODS и STOS можно прекрасно использовать вместе для копирования буферов. Например, следующая подпрограмма копирует завершающуюся нулевым символом строку, записанную по адресу DS:SI, в строку по адресу ES:DI:

```
;
; Подпрограмма для копирования завершающейся нулем строки
; в другую строку
;
; Ввод:
; DS:SI - строка, из которой выполняется копирование
; ES:DI - строка, в которую выполняется копирование
;
; Вывод: нет
;
; Изменяемые регистры: AL, SI, DI
;
CopyString PROC
 cld ; обеспечить увеличение SI и
 ; DI в строковых инструкциях
CopyStringLoop:
 lodsb ; получить символ исходной
 ; строки
 stosb ; записать символ в выходную
 ; строку
 cmp al,0 ; последним символом строки
 ; был 0?
 jnz CopyStringLoop ; нет, обработать следующий символ
 ret ; да, выполнено
CopyString ENDP
```

Аналогично вы можете использовать инструкции LODS и STOS для копирования блока байт, которые не завершаются нулем, используя для этого цикл:

```
mov cx,ARRAY_LENGTH_IN_WORDS ; размер массива
mov si,OFFSET SourceArray ; исходный массив
```

```
mov ax,SEG SourceArray
mov dx,ax
mov di,OFFSET DestArray ; целевой массив
mov ax,SEG DestArray
mov es,ax
cld
CopyLoop:
lodsw
stosw
loop CopyLoop
```

Однако для перемещения байта или слова из одного места в памяти в другое есть еще более лучший способ. Это инструкция MOVS.

3.2.3 Инструкция MOVS

Инструкция MOVS аналогична инструкциям LODS и STOS, если их объединить в одну инструкцию. Эта инструкция считывает байт или слово, записанное по адресу DS:SI, а затем записывает это значение по адресу, определяемому парой регистров ES:DI. Слово или байт не передается при этом через регистры, поэтому содержимое регистра AX не изменяется. Инструкция MOVSB имеет минимально возможную для инструкции длину. Она занимает только один байт, а работает еще быстрее, чем комбинация инструкций LODS и STOS. С применением инструкции MOVS последний пример приобретает вид:

```
mov cx,ARRAY_LENGTH_IN_WORDS
mov si,OFFSET SourceArray
mov ax,SEG SourceArray
mov ds,ax
mov di,OFFSET DestArray
mov ax,SEG DestArray
mov es,ax
cld
CopyLoop:
movsw
loop CopyLoop
```

3.2.4 Повторение строковой инструкции

Хотя в последнем примере код выглядит довольно эффективным, неплохо было бы избавиться от инструкции LOOP и перемещать весь массив с помощью одной инструкции. Инструкции процессора 8086 предоставляют такую возможность. Это форма строковых инструкций с префиксом REP.

Префикс повторения REP - это не инструкция, а префикс инструкции. Префикс инструкции изменяет работу последующей инструкции. Префикс REP делает следующее: он указывает, что последующую инструкцию нужно повторно выполнять до тех пор, пока содержимое регистра CX не станет равным 0. (Если регистр CX равен 0 в начале выполнения инструкции, то инструкция выполняется 0 раз, другими словами, никаких действий не производится.)

Используя префикс REP, можно заменить в последнем примере инструкции:

```
CopyLoop:
movsw
loop CopyLoop
```

на инструкцию:

```
rep movsb
```

Эта инструкция будет перемещать блок из 65535 слов (0FFFFh) из памяти, начинающейся с адреса DS:SI, в память, начинающуюся с адреса, определяемого регистрами ES:DI.

Конечно, для выполнения инструкции 65535 раз потребуется гораздо больше времени, чем для выполнения инструкции один раз, ведь для обращения ко всей этой памяти требуется время. Однако каждое повторение (с помощью префикса) строковой инструкции выполняется быстрее, чем выполнение одной строковой инструкции. Это позволяет получить очень быстрый способ чтения из памяти, записи в память и копирования.

Префикс REP можно использовать не только с инструкцией MOVS, но также и с инструкциями LODS и STOS (и инструкциями SCAS и CMPS - это мы обсудим позднее). Инструкцию STOS можно с успехом повторять для очистки или заполнения блоков памяти, например:

```
cld
mov ax,SEG WordArray
mov es,ax
mov di,OFFSET WordArray
sub ax,ax
mov cx,WORD_ARRAY_LENGTH
rep stosw
```

Здесь массив WordArray заполняется нулями. Для повторения инструкции LODS соответствующее полезное приложение придумать трудно.

Префикс REP вызывает повторение только строковой инструкции. Инструкция типа:

```
rep mov ax,[bx]
```

не имеет смысла. В этом случае префикс REP игнорируется и выполняется инструкция:

```
mov ax,[bx]
```

3.2.4 Сравнение строк

Команда CMPS сравнивает содержимое одной области памяти (адресуемой регистрами DS:SI) с содержимым другой области (адресуемой как ES:DI). В зависимости от флага DF команда CMPS также увеличивает или уменьшает адреса в регистрах SI и DI на 1 для байта или на 2 для слова. Команда CMPS устанавливает флаги AF, CF, OF, PF, SF и ZF. При использовании префикса REP в регистре CX должна находиться длина сравниваемых полей. Команда CMPS может сравнивать любое число байт или слов. Рассмотрим процесс сравнения двух строк, содержащих имена JEAN и JOAN. Сравнение побайтно слева направо приводит к следующему:

```
J : J Равно
E : O Не равно (E меньше O)
A : A Равно
N : N Равно
```

Сравнение всех четырех байт заканчивается сравнением N:N - равно/нуль. Так как имена "не равны", операция должна прекратиться, как только будет обнаружено условие "не равно". Для этих целей команда REP имеет модификацию REPE, которая повторяет сравнение до тех пор, пока сравниваемые элементы равны, или регистр CX не равен нулю. Кодируется повторяющееся однобайтовое сравнение следующим образом:

```
REPE CMPSB
```

3.3 Режимы адресации к памяти

Как при использовании операнда в памяти задать ту ячейку памяти, с которой вы хотите работать? Очевидный ответ состоит в том, чтобы присвоить нужной переменной в памяти имя (как мы это делали в последнем разделе). С помощью, например, следующих операторов вы можете вычесть переменную памяти Debts (долги) из переменной памяти Assets (имущество):

```
Assets DW ?
Debts DW ?
.
.
.
mov ax,[Debts]
sub [Assets],ax
```

Однако адресация к памяти имеет и более глубокий смысл, который не бросается в глаза. Предположим, у вас имеется символьная строка с именем CharString, содержащая буквы ABCDEFGHIGKLM, которые начинаются в сегменте данных со смещения 100, как показано на Рис. 1. Каким образом можно считать

девятый символ (I), который расположен по адресу 108? В языке Си вы можете просто использовать оператор:

```
C = CharString[8];
```

(в Си элементы нумеруются с 0), а в Паскале:

```
C := CharString[9];
```

Как же это можно сделать в Ассемблере? Прямая ссылка на строку CharString здесь, конечно, не подходит, так как первым символом является символ A.

```
 99 | ? |
 |--------------|
 CharString ---> 100 | 'A' |
 |--------------|
 101 | 'B' |
 |--------------|
 102 | 'C' |
 |--------------|
 103 | 'D' |
 |--------------|
 104 | 'E' |
 |--------------|
 105 | 'F' |
 |--------------|
 106 | 'G' |
 |--------------|
 107 | 'H' |
 |--------------|
 108 | 'I' |
```

|----------|

109 | 'J' |

|----------|

110 | 'K' |

|----------|

111 | 'L' |

|----------|

112 | 'M' |

|----------|

113 | 0 |

|----------|

114 | ? |

|----------|

. .

. .

Рисунок 1 – Ячейки памяти со строкой символов CharString.

В действительности язык Ассемблера обеспечивает несколько различных способов адресации к строкам символов, массивам и буферам данных. Наиболее простой способ состоит в том, чтобы считать девятый по счету символ строки CharString:

```
.DATA
CharString DB 'ABCDEFGHIJKLM'
.
.
.
.CODE
.
.
.
mov ax,@Data
mov ds,ax
```

```
mov al,[CharString+8]
```

В данном случае это то же самое, что:

```
mov al,[100+8]
```

так как CharString начинается со смещения 100. Все, что заключено в квадратные скобки, интерпретируется Турбо Ассемблером, как адрес; поэтому смещение CharString и 8 складывается и используется в качестве адреса памяти. Инструкция принимает вид:

```
mov al,[108]
```

как показано на Рис. 2.

```
 . .
 . .
 | |
 |--------------|
 99 | ? |
 |--------------|
CharString ------> 100 | 'A' |
 |--------------|
 101 | 'B' |
 |--------------|
 102 | 'C' |
 |--------------|
 103 | 'D' |
 |--------------|
 104 | 'E' |
 |--------------|
 105 | 'F' |
```

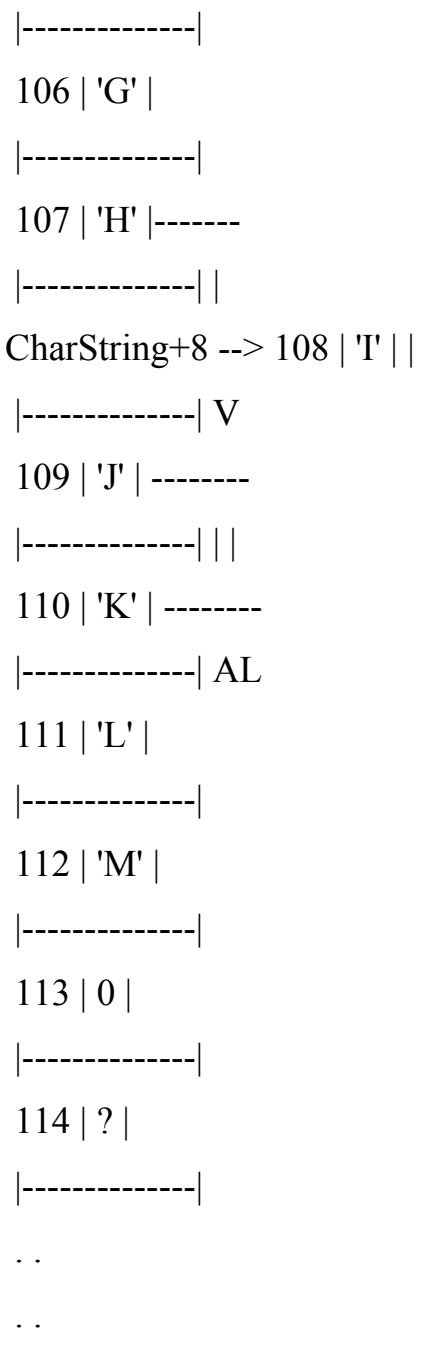


Рисунок 2 – Адресация строки символов строки CharString.

Такой тип адресации, когда ячейка памяти задается ее именем, плюс некоторая константа, называется непосредственной (прямой) адресацией. Хотя непосредственная адресация - это хороший метод, она не отличается достаточной гибкостью, поскольку обращение выполняется каждый раз по одному и тому же адресу памяти. Поэтому давайте рассмотрим другой, более гибкий путь адресации памяти.

Рассмотрим следующий фрагмент программы, где в регистр AL также загружается девятый символ CharString:

```
mov bx,OFFSET CharString+8
mov al,[bx]
```

В данном примере для ссылки на девятый символ используется регистр BX. Первая инструкция загружает в регистр BX смещение CharString (вспомните о том, что операция OFFSET возвращает смещение метки в памяти), плюс 8. (Вычисление OFFSET и сложение для этого выражения выполняется Турбо Ассемблером во время ассемблирования.) Вторая инструкция определяет, что AL нужно сложить с содержимым по смещению в памяти, на которое указывает регистр BX (см. Рис. 3).

```
mov al,[108]
```

как показано на рисунке 3.

```
 . .
 . .
 | |
 |--------------|
 99 | ? |
 |--------------|
CharString ------> 100 | 'A' |
 |--------------|
 101 | 'B' |
 |--------------|
 102 | 'C' |
 |--------------|
 103 | 'D' |
 |--------------|
 104 | 'E' |
```

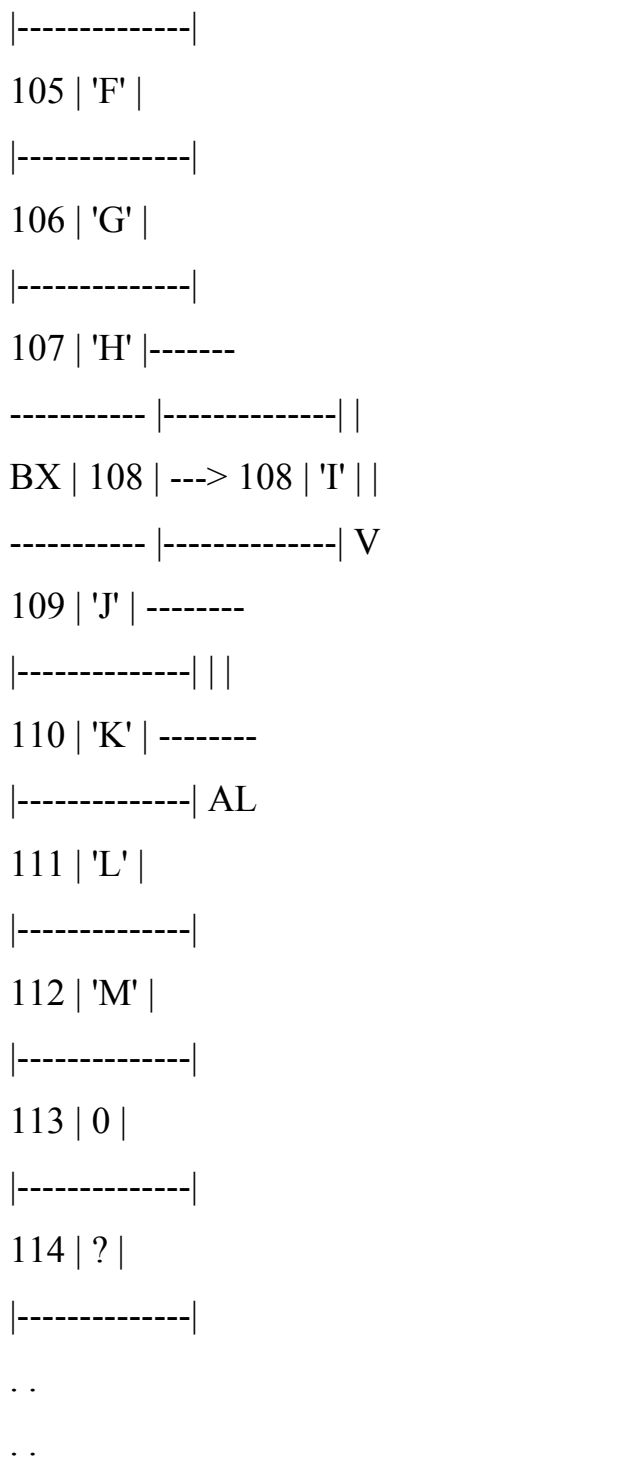


Рисунок 3 – Использование регистра BX для адресации к строке CharString

Квадратные скобки показывают, что в качестве операнда-источника должна быть использоваться ячейка, на которую указывает регистр BX, а не сам регистр BX. Не забывайте указывать квадратные скобки при использовании BX в качестве указателя памяти. Например:

```
mov ax,[bx] ; загрузить AX из ячейки памяти,
; на которую указывает BX
```

и

```
mov ax,bx ; загрузить в AX содержимое
; регистра BX
```

это две совершенно различные инструкции.

Может возникнуть вопрос, зачем сначала загружать в регистр BX смещение ячейки памяти и затем использовать BX, как указатель, если тоже самое можно сделать с помощью одной инструкции с непосредственным операндом? Особое свойство регистров, используемых в качестве указателей, состоит в том, что в отличие от инструкций, использующих непосредственные операнды, инструкции, использующие в качестве указателей регистры, могут ссылаться в разное время (в процессе выполнения программы) на разные ячейки памяти.

Предположим, вы хотите определить последний символ завершающейся нулем строки CharString. аЧтобы это сделать, вы должны, начиная с первого символа строки CharString, найти завершающий строку нулевой байт, затем вернуться назад на один символ и считать этот последний символ. Это невозможно сделать с помощью непосредственной адресации, так как строка может иметь произвольную длину. Использование регистра BX значительно облегчает задачу:

```
mov bx,OFFSET CharString ; указывает на строку
FindLastCharLoop:
mov al,[bx] ; получить следующий
; символ строки
cmp al,0 ; это нулевой байт?
je FoundEndOfString ; да, вернуться на
; один символ
inc bx
jmp FilnLastCharLoop ; проверить следующий
; символ
FoundEndOfString:
```

```
dec bx ; снова указывает на
; последний символ
mov al,[bx] ; получить последний
. ; символ строки
```

Если вы собираетесь выполнять в памяти поиск символов или слов, работать с массивами, или копировать блоки данных, вы поймете, что использование регистров-указателей дает неоценимую помощь.

BX - это не единственный регистр, который можно использовать для ссылки на память. Допускается также использовать вместе с необязательным значением-константой или меткой регистры BP, SI и DI. Общий вид операндов в памяти выглядит следующим образом:

```
[базовый регистр + индексный регистр + смещение]
```

где базовый регистр - это BX или BP, индексный регистр - SI или DI, а смещение - любая 16-битовая константа, включая метки и выражения. Каждый раз, когда выполняется инструкция, использующая операнд в памяти, процессором 8086 эти три компоненты складываются. Каждая из трех частей операнда в памяти является необязательной, хотя (это очевидно) вы должны использовать один из трех элементов, иначе вы не получите адреса в памяти. Вот как элементы операнда в памяти выглядят в другом формате:

```
BX SI
или + или + Смещение
BP DI
(база) (индекс)
```

Существует 16 способов задания адреса в памяти:

```
[смещение] [bp+смещение]
[bx] [bx+смещение]
[si]    [si+смещение]
[di]    [di+смещение]
[bx+si]        [bx+si+смещение]
[bx+di [bx+di+смещение]
```

```
[bp+si] [bp+si+смещение]
[bp+di] [bp+di+смещение]
```

где смещение - это то, что можно свести к 16-битовому постоянному значению.

Поведение регистра BP несколько отличается от регистра BX. Вспомните, что в то время как регистр BX используется, как смещение внутри сегмента данных, регистр BP используется, как смещение в сегменте стека. Это означает, что регистр BP не может обычно использоваться для адресации к строке CharString, которая находится в сегменте данных. . В данный момент достаточно знать, что регистр BP можно использовать так же, как мы использовали в примерах регистр BX, только адресуемые данные должны в этом случае находиться в стеке.

(На самом деле регистр BP можно использовать и для адресации к сегменту данных, а BX, SI и DI - для адресации к сегменту стека, дополнительному сегменту или сегменту кода. Для этого используются префиксы переопределения сегментов (segment override prefixes). О некоторых из них мы расскажем в Главе 10. Однако в большинстве случаев они вам не понадобятся, поэтому пока мы просто забудем об их существовании.)

Наконец, квадратные скобки, в которые заключаются непосредственные адреса, являются необязательными. То есть инструкции:

```
mov al,[MemVar]
```

и

```
mov al,MemVar
```

выполняют одни и те же действия. Тем не менее мы настоятельно рекомендуем вам заключать все ссылки на память в квадратные скобки. Это поможет избежать путаницы и сделает вашу программу более ясной и понятной. Несомненно, вы столкнетесь с программами, в которых квадратные скобки отсутствуют, так как некоторые все же считают, что в таком виде программа воспринимается лучше. Это,

в общем, дело вкуса, но если вы выберете стиль адресации по одной ячейке памяти и будете содержательно его использовать, вам будет легче писать программы.

Вы можете использовать также такую форму адресации к памяти:

```
mov al,CharString[bx]
```

или даже

```
mov al,CharString[bx][si]+1
```

Все эти формы представляют собой то же самое, что размещение отдельных элементов адресации к памяти в одной паре квадратных скобок и разделение их знаком плюс. Таким образом, последний оператор эквивалентен оператору:

```
mov al,[charString+bx+si+1]
```

Здесь снова нужно выбрать ту форму записи, которая вам больше нравится, и придерживаться ее.

Квадратные скобки, в которые заключаются регистры, указывающие на ячейки памяти, являются обязательными. Без этих скобок, BX, например, интерпретируется, как операнд, а не как ссылка на операнд.

3.4 Ввод-вывод

Использовать базовые DOS для ввода и вывода строк крайне неудобно т.к. они обеспечивают только посимвольную обработку данных. И для ввода с клавиатуры одной строки нам пришлось бы организовывать цикл. Существует более эффективные способы обработки строк.

Рассмотрим пример использования команд условных переходов для обработки символов. Пусть мы вводим с клавиатуры некоторую строку символов (например, имя файла), и хотим, чтобы в программе эта строка была записана прописными буквами, независимо от того, какие буквы использовались при ее вводе. Между

прочим, при вводе с клавиатуры команд DOS система всегда выполняет эту операцию, поэтому и команды, и ключи, и имена файлов можно вводить как прописными, так и строчными буквами - DOS во всех случаях преобразует все буквы в прописные.

```
code segment
assume cs:code, ds:data
main proc
        mov ax,data             ;Инициализируем
        mov ds,ax               ;регистр DS
;Выведем служебное сообщение
        mov ah,09h              ;Функция вывода
        mov dx,offset msg ;адрес сообщения
        int 21h
;Поставим запрос к DOS на ввод строки
        mov ah,3Fh              ;функция ввода
        mov bx,0                ;дескриптор клавиатуры
        mov cx,80               ;ввод максимум 80 байт
        mov dx,offset buf  ;адрес буфера ввода
        int 21h
        mov actlen,ax           ;фактически введено
;Превратим строчные русские буквы в прописные
        mov cx,actlen           ;длина введенной строки
        mov si,0                ;указатель в буфере
filter: mov al,buf[si]      ;возьмем символ
        cmp al,'а'              ;меньше «а»?
        jb noletter             ;да, не преобразовывать
        cmp al,'я'              ;Больше «я»?
        ja noletter             ;да, не преобразовывать
        cmp al,'п'              ;больше «п»?
        ja more                 ;да, на дальнейшую программу
        sub al,20h              ;«а»..«п». преобразуем в прописную букву
        jmp store               ;на сохранение в буфер
more: cmp al,'р'              ;меньше «р»(псевдографика)?
        jb noletter             ;>п,<р не изменять
        sub al,50h              ; «р»..«я». Преобразуем в прописную букву
store: mov buf[si],al       ;отправим назад в buf
noletter:       inc si          ;сместим указатель
loop filter                     ;цикл по всем символам
;Выведем результат преобразования на экран
        mov ah,40h              ;функция ввода
        mov bx,1                ;дескриптор экрана
        moc cx,actlen           ;длина сообщения
```

```
        mov dx,offset buf  ;адрес сообщения
        int 21h
        mov ah,01                  ;остановим программу
        int 21h                    ;в ожидании нажатия клавиши
 ;Завершим программу
        mov ax4C00h
        int 21h
main endp
code ends
data segment
msg db 'Вводите! $'
buf db 80 dup(' ')
actlen dw 0
data ends
stk segment stack
dw 128 dup (?)
stk ends
end main
```

В начале программы на экран выводится служебное сообщение "Вводите!", которое служит запросом программы, адресованным пользователю. Далее с помощью функции DOS 3Fh выполняется ввод строки текста с клавиатуры. Функция 3Fh может вводить данные из разных устройств - файлов, последовательного порта, клавиатуры.

Различные устройства идентифицируются их дескрипторами. При работе с файлами дескриптор каждого файла создается системой в процессе операции открытия или создания этого файла, а для стандартных устройств - клавиатуры, экрана, принтера и последовательного порта действуют дескрипторы, закрепляемые за этими устройствами при загрузке системы. Для ввода с клавиатуры используется дескриптор 0, для вывода на экран дескриптор.

При вызове функции 3Fh в регистр BX следует занести требуемый дескриптор, в регистр DX - адрес области в программе, выделенной для приема вводимых с клавиатуры символов, а в регистр CX - максимальное число вводимых символов. Мы считаем, что пользователь не будет вводить более 80 символов. Можно ввести и меньше; в любом случае ввод строки следует завершить нажатием клавиши <Enter>. Функция 3Fh, отработав, вернет в регистре AX реальное число введенных символов

(включая коды 13 и 10, образуемые при нажатии клавиши <Enter>). В примере 3.5 число введенных символов сохраняется в ячейке actlen с целью использования далее по ходу программы.

Далее в цикле из actlen шагов выполняется анализ каждого введенного символа путем сравнения с границами диапазонов строчных русских букв. Русские строчные буквы размещаются в двух диапазонах кодов ASCII (а...п и р...с), причем для преобразования в прописные букв первого диапазона их код следует уменьшать на 20h, а для преобразования букв второго диапазона - на 50h. Поэтому анализ проводится с помощью четырех команд сравнения cmp и соответствующих команд условных переходов. Модифицированный символ записывается на то же место в буфере buf.

После завершения анализа и преобразования введенных символов, выполняется контрольный вывод содержимого buf на экран. Поскольку мы заранее не знаем, сколько символов будет введено, вывод на экран осуществляется функцией 40h, среди параметров которой указывается число выводимых символов. Так же, как и в случае функции ввода 3Fh, для функции вывода 40h в регистре BX необходимо указать дескриптор устройства ввода, в данном случае экрана, а в регистре DX - адрес выводимой строки.

3.6 Задания

Во всех вариантах необходимо реализовать программу работы со строками. Исходная строка вводится с клавиатуры, результат выводится на экран. Слова в строке могут быть разделены пробелами и знаками препинания.

1) Найти слова, начинающиеся на заданную с клавиатуры букву, и перевернуть.
2) Найти слова, оканчивающиеся на заданную трехбуквенную комбинацию.
3) Отсортировать строку по длине слов.
4) Переставить слова – 6 слов: 1-6, 2-5, 3-4; 7 слов: 1-7, 2-6, 3-5, 4 на месте (цифры словами не считаются).

5) Подсчитать в строке количество слов, содержащих в середине гласную букву.
6) Отсортировать слова строки по третьей букве.
7) Найти слова, в которых больше трех повторяющихся символов.
8) Подсчитать количество слов-перевертышей в строке (шалаш).
9) Отсортировать слова в строки по алфавиту (по 1-ой букве, потом по 2-ой и т.д.).
10) Удалить в каждом слове строки повторяющиеся в нем буквы.
11) Вставить в каждое слово строки после заданного символа символ, введенный пользователем.
12) Найти слова, оканчивающиеся на заданную с клавиатуры букву, и перевернуть.
13) В каждом слове строки удвоить буквы.
14) Отсортировать буквы в каждом слове по алфавиту, оставляя позицию данного слова в строке неизменной.
15) Заменить все слова-перевертыши строки словом, введенным пользователем.
16) Удалить из каждого слова строки все гласные и вывести те гласные, которых не было в строке.
17) В каждом слове строки заменить 1-ую букву на последнюю, 2-ую на предпоследнюю и т.д.
18) Отсортировать слова в строке по количеству разных согласных букв, встречающихся в них.

4 РАБОТА С МАССИВАМИ И СТЕКОМ НА ЯЗЫКЕ АССЕМБЛЕРА

4.1 Общие сведения о массивах

Как структура представления, массив является упорядоченным множеством элементов определенного типа. Упорядоченность массива определяется набором целых чисел, называемых индексами, которые связываются с каждым элементом массива и однозначно конкретизируют его расположение среди других элементом массива. Локализация конкретного элемента массива - ключевая задача при разработке любых алгоритмов, работающих с массивами.

Наиболее просто представляются одномерные массивы. Соответствующая им структура хранения — это вектор. Она однозначна и есть не что иное, как просто последовательное расположение элементов в памяти. Чтобы локализовать нужный элемент одномерного массива, достаточно знать его индекс. Так как ассемблер не имеет средств для работы с массивом как структурой данных, то для доступа к элементу массива необходимо вычислить его адрес.

Представление двумерных массивов немного сложнее. Здесь мы имеем случай, когда структуры хранения и представления различны. О структуре представления говорить излишне — это матрица. Структура хранения остается прежней — вектор. Но теперь его нельзя без специальных оговорок интерпретировать однозначно. Все зависит от того, как решил разработчик программы «вытянуть» массив — по строкам или по столбцам. Наиболее естествен порядок расположения элементов массива — по строкам. При этом наиболее быстро изменяется последний элемент индекса.

4.2 Ввод – вывод массива

Пример ввода массива:

```
Read Proc
mov dl,10
mov ah,02h
int 21h
```

```
mov dl,13
mov ah,02h
int 21h ; перевод строки
mov ah,0ah
lea dx,len
int 21h         ; считали данные в len
mov ch,0
mov cl,len+1
mov si,offset val
mov bl,10
mov ax,0
L1:
mul bl
mov dl,[si]
sub dl,30h
mov dh,0
add ax,dx
inc si
LOOP L1 ; запускаем цикл
ret
Read EndP
```

Пример вывода массива:

```
Output Proc          ; вывод массива
mov bx,0
mov di,-1
n2:
inc pp
mov dh,pp
mov ah,02
mov bh,00
mov dl,00
int 10h
n1: inc di
mov dx,offset sr
mov ah,09h
int 21h
mov al,mas[bx][di]
PUSH di
xor ah,ah
mov kx,ax
viv:
```

```
 cmp al,0
 jge nosing
 neg al
 mov kx,ax
 mov dl,'-'
 mov ax,0200h
 int 21h
nosing:
 xor si,si
 xor dx,dx
 mov di,10
 mov ax,kx
iter1:
 div di
 xor dl,'0'
 mov str2[si],dl
 xor dx,dx
 inc si
 cmp ax,0
 ja iter1
 mov cx,si
 dec si
 mov ax,0200h
iter2:
 MOV dl,str2[si]
 int 21h
 dec si
 loop iter2
POP di
cmp di,jx
jb n1
mov di,-1
;Bi
Call OutBi
cmp bx,ix
jb n2
ret
Output EndP
```

4.3 Способы сортировки массивов.

Метод пузырька

```
Процедура bubble_sort
; сортирует массив слов методом пузырьковой сортировки
```

```
; ввод: DS:DI = адрес массива
; DX = размер массива (в словах)
bubble_sort proc near
 pusha
 cld
 cmp dx,1
 jbe sort_exit ; выйти, если сортировать нечего
 dec dx
sb_loop1:
 mov cx,dx ; установить длину цикла
 xor bx,bx ; BX будет флагом обмена
 mov si,di ; SI будет указателем на
 ; текущий элемент
sn_loop2:
 lodsw ; прочитать следующее слово
 cmp ax,word ptr [si]
 jbe no_swap ; если элементы не
 ; в порядке,
 xchg ax,word ptr [si] ; поменять их местами
 mov word ptr [si-2],ax
 inc bx ; и установить флаг в 1,
no_swap:
 loop on_loop2
 cmp bx,0 ; если сортировка не закончилась,
 jne sn_loop1 ; перейти к следующему элементу
sort_exit:
 popa
 ret
bubble_sort endp
```

Пузырьковая сортировка осуществляется так медленно потому, что сравнения выполняются лишь между соседними элементами. Чтобы получить более быстрый метод сортировки перестановкой, следует выполнять сравнение и перестановку элементов, отстоящих далеко друг от друга. На этой идее основан алгоритм, который называется «быстрая сортировка». Он работает следующим образом: делается предположение, что первый элемент является средним по отношению к остальным. На основе такого предположения все элементы разбиваются на две группы - больше и меньше предполагаемого среднего. Затем обе группы отдельно сортируются таким же методом. В худшем случае быстрая сортировка массива из N элементов требует N^2 операций, но в среднем случае - только $2n*\log_2 n$ сравнений и еще меньшее число перестановок.

Метод быстрой сортировки:

```
; Процедура quick_sort
; сортирует массив слов методом быстрой сортировки
; ввод: DS:BX = адрес массива
; DX = число элементов массива
quicksort proc near
cmp dx,1 ; Если число элементов 1 или 0,
jle qsort_done ; то сортировка уже закончилась
xor di,di ; индекс для просмотра сверху (DI = 0)
mov si,dx ; индекс для просмотра снизу (SI = DX)
dec si ; SI = DX-1,          так как элементы нумеруются с нуля,
shl si,1 ; и умножить на 2,                 так как это массив слов
mov ax,word ptr [bx] ; AX = элемент X1, объявленный средним
step_2: ; просмотр массива снизу, пока не встретится
 ; элемент, меньший или равный X1
cmp word ptr [bx][si],ax ; сравнить XDI и X1
jle step_3      ; если XSI больше,
sub si,2        ; перейти к следующему снизу элементу
jmp short step_2     ; и продолжить просмотр
step_3:                         ; просмотр массива сверху, пока не встретится
 ; элемент меньше X1              или оба просмотра не придут
 ; в одну точку
cmp si,di ; если просмотры встретились,
je step_5               ; перейти к шагу 5,
add di,2 ; иначе: перейти
                        ; к следующему сверху элементу,
cmp word ptr [bx][di],ax          ; если он меньше X1,
jl step_3               ; продолжить шаг 3
step_4:
; DI указывает на элемент, который не должен быть
; в верхней части, SI указывает на элемент,
; который не должен быть в нижней. Поменять их местами
mov cx,word ptr [bx][di] ; CX = XDI
xchg cx,word ptr [bx][si] ; CX = XSI, XSI = XDI
mov word ptr [bx][di],cx ; XDI = CX
jmp short step_2
step_5: ; Просмотры встретились. Все элементы в нижней
 ; группе больше X1, все элементы в верхней группе
 ; и текущий - меньше или равны X1 Осталось
 ; поменять местами X1 и текущий элемент:
xchg ax,word ptr [bx][di] ; AX = XDI, XDI = X1
mov word ptr [bx],ax ; X1 = AX
; теперь можно отсортировать каждую из полученных групп
push dx
```

```
push di
push bx

mov dx,di ; длина массива X1...XDI-1
 shr dx,1 ; в DX
 call quick_sort ; сортировка

 pop bx
 pop di
 pop dx
 add bx,di ; начало массива XDI+1...XN
 add bx,2 ; в BX
 shr di,1 ; длина массива XDI+1...XN
 inc di
 sub dx,di ; в DX
 call quicksort ; сортировка
qsort_done: ret
quicksort endp
```

Кроме того, что быстрая сортировка - самый известный пример алгоритма, использующего рекурсию, то есть вызывающего самого себя. Это еще и самая быстрая из сортировок «на месте», то есть сортировка, использующая только ту память, в которой хранятся элементы сортируемого массива. Можно доказать, что сортировку нельзя выполнить быстрее, чем за $n*\log_2 n$ операций, ни в худшем, ни в среднем случаях; и быстрая сортировка достаточно хорошо приближается к этому пределу в среднем случае. Сортировки, достигающие теоретического предела, тоже существуют — это сортировки турнирным выбором и сортировки вставлением в сбалансированные деревья, но для их работы требуется резервирование дополнительной памяти Так что, например, работа со сбалансированными деревьями будет происходить медленно из-за дополнительных затрат на поддержку сложных структур данных в памяти.

4.4 Работа со стеком в ассемблере

4.4.1 Команды работы со стеком

В процессорах Intel команду BSWAP можно использовать и для обращения порядка байт в 16-битных регистрах, но в некоторых совместимых процессорах других фирм этот вариант BSWAP не реализован.

Команда:	**PUSH** источник
Назначение:	Поместить данные в стек
Процессор:	8086

Команда помещает содержимое источника в стек. В качестве параметра «источник» может быть регистр, сегментный регистр, непосредственный операнд или переменная. Фактически эта команда копирует содержимое источника в память по адресу SS:[ESP] и уменьшает ESP на размер источника в байтах (2 или 4). Команда PUSH практически всегда используется в паре с POP (считать данные из стека). Так, например, чтобы скопировать содержимое одного сегментного регистра в другой (что нельзя выполнить одной командой MOV), можно использовать такую последовательность команд:

```
push cs
pop ds ; теперь DS указывает на тот же сегмент, что и CS
```

Другое частое применение команд PUSH/POP — временное хранение переменных, например:

```
push ax ; сохраняет текущее значение AX
... ; здесь располагаются какие-нибудь команды,
; которые используют AX, например CMPXCHG
pop ax ; восстанавливает старое значение AX
```

Начиная с 80286, команда PUSH ESP (или SP) помещает в стек значение ESP до того, как эта же команда его уменьшит, в то время как на 8086 SP помещался в стек уже уменьшенным на два.

Команда:	**POP** приемник

Назначение:	Считать данные из стека
Процессор:	8086

Команда помещает в приемник слово или двойное слово, находящееся в вершине стека, увеличивая ESP на 2 или 4 соответственно. POP выполняет действие, полностью обратное PUSH. Приемником может быть регистр общего назначения, сегментный регистр, кроме CS (чтобы загрузить CS из стека, надо воспользоваться командой RET), или переменная. Если в роли приемника выступает операнд, использующий ESP для косвенной адресации, команда POP вычисляет адрес операнда уже после того, как она увеличивает ESP.

Команда:	**PUSHA** **PUSHAD**
Назначение:	Поместить в стек все регистры общего назначения
Процессор:	80186 80386

PUSHA помещает в стек регистры в следующем порядке: AX, CX, DX, BX, SP, BP, SI и DI. PUSHAD помещает в стек EAX, ECX, EDX, EBX, ESP, EBP, ESI и EDI. (В случае SP и ESP используется значение, которое находилось в этом регистре до начала работы команды.) В паре с командами POPA/POPAD, считывающими эти же регистры из стека в обратном порядке, это позволяет писать подпрограммы (обычно обработчики прерываний), которые не должны изменять значения регистров по окончании своей работы. В начале такой подпрограммы вызывают команду PUSHA, а в конце — POPA.

На самом деле PUSHA и PUSHAD — одна и та же команда с кодом 60h. Ее поведение определяется тем, выполняется ли она в 16- или в 32-битном режиме. Если программист использует команду PUSHAD в 16-битном сегменте или PUSHA в 32-битном, ассемблер просто записывает перед ней префикс изменения размерности операнда (66h).

Это же будет распространяться на некоторые другие пары команд: POPA/POPAD, POPF/POPFD, PUSHF/PUSHFD, JCXZ/JECXZ, CMPSW/CMPSD, INSW/INSD, LODSW/LODSD, MOVSW/MOVSD, OUTSW/OUTSD, SCASW/SCASD и STOSW/STOSD.

Команда:	**POPA** **POPAD**
Назначение:	Загрузить из стека все регистры общего назначения
Процессор:	80186 80386

4.4.2 Передача параметров в стеке

Параметры помещаются в стек сразу перед вызовом процедуры. Именно этот метод используют языки высокого уровня, такие как C и Pascal. Для чтения параметров из стека обычно используют не команду POP, а регистр BP, в который помещают адрес вершины стека после входа в процедуру:

```
 push parameter1 ; поместить параметр в стек
 push parameter2
 call procedure
 add sp,4 ; освободить стек от параметров
 [...]
procedure proc near
 push bp
 mov bp,sp
(команды, которые могут использовать стек)
 mov ax,[bp+4] ; считать параметр 2.
; Его адрес в сегменте стека BP + 4, потому что при выполнении
; команды CALL в стек поместили адрес возврата - 2 байта для процедуры
; типа NEAR (или 4 - для FAR), а потом еще и BP - 2 байта
 mov bx,[bp+6] ; считать параметр 1
(остальные команды)
 pop bp
 ret
procedure endp
```

Параметры в стеке, адрес возврата и старое значение BP вместе называются активационной записью функции.

Для удобства ссылок на параметры, переданные в стеке, внутри функции иногда используют директивы EQU, чтобы не писать каждый раз точное смещение параметра от начала активационной записи (то есть от BP), например так:

```
push X
```

```
 push Y
 push Z
 call xyzzy
 [...]
xyzzy proc near
xyzzy_z equ [bp+8]
xyzzy_y equ [bp+6]
xyzzy_x equ [bp+4]
 push bp
 mov bp,sp
(команды, которые могут использовать стек)
 mov ax,xyzzy_x ;считать параметр X
(остальные команды)
 pop bp
 ret 6
xyzzy endp
```

При внимательном анализе этого метода передачи параметров возникает сразу два вопроса: кто должен удалять параметры из стека, процедура или вызывающая ее программа, и в каком порядке помещать параметры в стек. В обоих случаях оказывается, что оба варианта имеют свои «за» и «против», так, например, если стек освобождает процедура (командой RET число_байтов), то код программы получается меньшим, а если за освобождение стека от параметров отвечает вызывающая функция, как в нашем примере, то становится возможным вызвать несколько функций с одними и теми же параметрами просто последовательными командами CALL. Первый способ, более строгий, используется при реализации процедур в языке Pascal, а второй, дающий больше возможностей для оптимизации, - в языке C. Разумеется, если передача параметров через стек применяется и для возврата результатов работы процедуры, из стека не надо удалять все параметры, но популярные языки высокого уровня не пользуются этим методом. Кроме того, в языке C параметры помещают в стек в обратном порядке (справа налево), так что становятся возможными функции с изменяемым числом параметров (как, например, printf - первый параметр, считываемый из [BP+4], определяет число остальных параметров). Но подробнее о тонкостях передачи параметров в стеке рассказано далее, а здесь приведен обзор методов.

4.4.3 Передача параметров в потоке кода

В этом необычном методе передаваемые процедуре данные размещаются прямо в коде программы, сразу после команды CALL (как реализована процедура print в одной из стандартных библиотек процедур для ассемблера UCRLIB):

```
call print
db "This ASCIZ-line will be printed",0
(следующая команда)
```

Чтобы прочитать параметр, процедура должна использовать его адрес, который автоматически передается в стеке как адрес возврата из процедуры. Разумеется, функция должна будет изменить адрес возврата на первый байт после конца переданных параметров перед выполнением команды RET. Например, процедуру print можно реализовать следующим образом:

```
print proc near
 push bp
 mov bp,sp
 push ax
 push si
 mov si,[bp+2]              ; прочитать адрес
                  ; возврата/начала данных
 cld              ; установить флаг направления
                  ; для команды lodsb
print_readchar:
 lodsb                  ; прочитать байт из строки,
 test al,al             ; если это 0 (конец строки),
 jz print_done                  ; вывод строки закончен
 int 29h                ; вывести символ в AL на экран
 jmp short print_readchar
print_done:
 mov [bp+2],si              ; поместить новый адрес возврата в стек
 pop si
 pop ax
 pop bp
 ret
print endp
```

Передача параметров в потоке кода, так же, как и передача параметров в стеке в обратном порядке (справа налево), позволяет передавать различное число параметров. Но этот метод - единственный, позволяющий передать по значению параметр различной длины, что и продемонстрировал этот пример. Доступ к параметрам, переданным в потоке кода, несколько медленнее, чем к параметрам,

переданным в регистрах, глобальных переменных или стеке, и примерно совпадает со следующим методом.

4.5 Задания

Дан двумерный массив. Размер массива и его элементы вводятся пользователем с клавиатуры. Результат работы программы выводится на экран.

Варианты:

1) Посчитать и вывести сумму положительных элементов, расположенных под главной диагональю матрицы.
2) Заменить элементы главной диагонали на суммы элементов соответствующих строк, затем отсортировать получившийся массив по возрастанию элементов главной диагонали.
3) Решить систему линейных уравнений методом Гаусса, в случае несуществования решения вывести сообщение.
4) Посчитать и вывести суммы элементов больше заданного пользователем числа, расположенных в верхнем и нижнем треугольниках, образуемых диагоналями. Заменить элементы кратные 3 левого треугольника на первую сумму, а кратные 5 на вторую.
5) Отсортировать массив по возрастанию элементов побочной диагонали, затем поменять местами элементы, расположенные над и под побочной диагональю.
6) Отсортировать строки матрицы по суммам модулей элементов и вывести столбец с максимальным количеством отрицательных элементов.
7) Подсчитать и вывести максимальный диагональный минор матрицы.
8) Отсортировать элементы матрицы по возрастанию среднего арифметического по столбцу.
9) Найти произведение матриц произвольного размера (должна производиться проверка возможности выполнения операции умножения).
10) Посчитать и вывести обратную матрицу методом Гаусса.

11) Найти в верхнем треугольнике, образуемом диагоналями максимальный элемент, а в нижнем минимальный. Затем увеличить минимальный элемент на 30%, а максимальный уменьшить на 30% от разности этих элементов. Повторять процедуру до тех пор, пока разность между элементами превышает 10%.
12) Найти и вывести определитель матрицы. Все элементы матрицы, значение которых больше определителя уменьшить на 20%, элементы, значение которых меньше увеличить на 30%.
13) В матрице поменять местами элементы треугольников, образованных диагоналями, заданное пользователем количество раз по часовой стрелке

    Пример:

    Исходная матрица

    1 2 3 4 5

    6 7 8 9 10

    11 12 13 14 15

    16 17 18 19 20

    21 22 23 24 25

    Результат (количество раз - 2)

    1 24 23 22 5

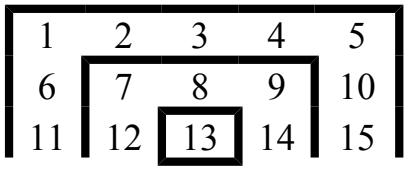
    20 7 18 9 6

    15 14 13 12 11

    10 17 8 19 16

    21 4 3 2 25

14) Повернуть элементы матрицы внутренний ряд по часовой стрелке, следующий против и т.д.



16	17	18	19	20
21	22	23	24	25

15) Найти максимум по каждому столбцу, затем вычесть из него n-1 (n количество элементов в массиве), к каждому элементу столбца прибавить 1. Найти минимум по строке, затем прибавить к нему n-1 (n количество элементов в массиве), из каждого элемента строки вычесть 1.
16) Сортировать каждый столбец по возрастанию, а строку по убыванию до тех пор, пока на очередном шаге итерации матрица останется неизменной, но не более 5 итераций.

5 РАБОТА С МАТЕМАТИЧЕСКИМ СОПРОЦЕССОРОМ В СРЕДЕ ASSEMBLER

5.1 Основные сведения

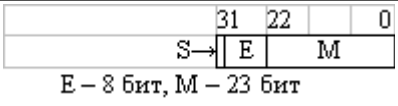
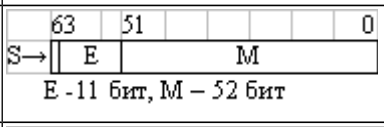
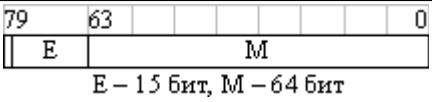
Сопроцессор - это специализированная интегральная схема, которая работает в содружестве с ЦП, но менее универсальна. Сопроцессор предназначен для выполнения специфического набора функций, например: выполнение операций с вещественными числами - математический сопроцессор, подготовка графических изображений и трехмерных сцен - графический сопроцессор, цифровая обработка сигналов - сигнальный сопроцессор и др.

Один из наиболее распространенных типов сопроцессоров - математический сопроцессор. Математический сопроцессор предназначен для быстрого выполнения арифметических операций с плавающей точкой, предоставления часто используемых вещественных констант (π, log210, log2e, ln2, :), вычисления тригонометрических и прочих трансцендентных функций (tg, arctg, log, ...).

Большинство современных математических сопроцессоров для представления вещественных чисел используют стандарт IEEE 754-1985 "IEEE1) Standard for Binary Floating-Point Arithmetics". Старший разряд двоичного представления вещественного числа всегда кодирует знак числа. Остальная часть разбивается на две части: экспоненту и мантиссу. Вещественное число вычисляется как: $(-1)^S \cdot 2^E \cdot M$, где S - знаковый бит числа, E - экспонента, M - мантисса. Если $1 \le M < 2$, то такое число называется нормализованным. При хранении нормализованных чисел сопроцессор отбрасывает целую часть мантиссы (она всегда 1), сохраняя лишь дробную часть. Экспонента кодируется со сдвигом на половину разрядной сетки, таким образом, удается избежать вопроса о кодировании знака экспоненты. Т.е. при 8-битной разрядности экспоненты код 0 соответствует числу -127, 1 - числу -126, ..., 255 числу +126 (экспонента вычисляется как код 127).

Стандарт IEEE-754 определяет три основных способа кодирования (типа) вещественных чисел.

Таблица 2 - Типы (способы кодирования) вещественных чисел

Тип	Диапазон значений (по модулю)	Двоичное представление
вещественное ординарной точности (single precision) - 32 бит	$1{,}18\cdot10^{-38}$... $3{,}40\cdot10^{38}$	
вещественное двойной точности (double precision) - 64 бит	$2{,}23\cdot10^{-308}$... $1{,}79\cdot10^{308}$	
вещественное расширенной точности (extended precision) - 80 бит	$3{,}37\cdot10^{-4932}$... $1{,}18\cdot10^{4392}$	

Если результат численной операции не может быть точно представлен в выбранном формате, сопроцессор выполняет округление в соответствии с полем RC (таблица 3). По умолчанию RC = 00.

Таблица 3 - Режимы округления сопроцессоров Intel x87

RC	Режим	Пример 1 $1{,}000E_21 < 2{,}23E_{10}0 < 1{,}001E_21$ $-1{,}001E_21 < -2{,}23E_{10}0 < -1{,}000E_21$	Пример 2 $1{,}000E_21 < 2{,}05E_{10}0 < 1{,}001E_21$ $-1{,}001E_21 < -2{,}05E_{10}0 < -1{,}000E_21$
00	Округление к ближайшему (или четному)	$2{,}23E_{10}0 \approx 1{,}001E_21$ $-2{,}23E_{10}0 \approx -1{,}001E_21$	$2{,}05E_{10}0 \approx 1{,}000E_21$ $-2{,}05E_{10}0 \approx -1{,}000E_21$
01	Округление вниз (к ∞)	$2{,}23E_{10}0 \approx 1{,}000E_21$ $-2{,}23E_{10}0 \approx -1{,}001E_21$	$2{,}05E_{10}0 \approx 1{,}000E_21$ $-2{,}05E_{10}0 \approx -1{,}001E_21$
10	Округление вверх (к $+\infty$)	$2{,}23E_{10}0 \approx 1{,}001E_21$ $-2{,}23E_{10}0 \approx -1{,}000E_21$	$2{,}05E_{10}0 \approx 1{,}001E_21$ $-2{,}05E_{10}0 \approx -1{,}000E_21$
11	Округление к нулю (усечение)	$2{,}23E_{10}0 \approx 1{,}000E_21$ $-2{,}23E_{10}0 \approx -1{,}000E_21$	$2{,}05E_{10}0 \approx 1{,}000E_21$ $-2{,}05E_{10}0 \approx -1{,}000E_21$

5.2 Команды сопроцессора

5.2.1 Команды пересылки данных

Команды загрузки в стек (Fpu LoaD):

- FLD - загружает из памяти в вершину стека ST(0) вещественное число

- FILD - загружает из памяти в вершину стека ST(0) целое число
- FBLD - загружает из памяти в вершину стека ST(0) двоично-десятичное число

Команды сохранения и извлечения из стека (Fpu STore and Pop):

- FSTP память - извлекает из вершины стека ST(0) в память вещественное число
- FISTP память - извлекает из вершины стека ST(0) в память целое число
- FBSTP память - извлекает из вершины стека ST(0) в память двоично-десятичное число

Эти команды сначала сохраняют вершину стека в памяти, а потом удаляют данные из вершины стека.

Команды копирования данных (Fpu STore):

- FST память - извлекает из вершины стека ST(0) в память вещественное число
- FIST память - извлекает из вершины стека ST(0) в память целое число
- FBST память - извлекает из вершины стека ST(0) в память двоично-десятичное число

Команда обмена (Fpu eXCHange):

- FXCH - обмен содержимым верхушки стека ST(0) и численного регистра, указанного в качестве операнда команды

5.2.2 Арифметические команды

Сопроцессор использует шесть основных типов арифметических команд:

- Fxxx

Первый операнд берется из верхушки стека (источник), второй - следующий элемент стека. Результат выполнения команды записывается в стек

- Fxxx память

Источник берется из памяти, приемником является верхушка стека ST(0). Указатель стека ST не изменяется, команда действительна только для операндов с одинарной и двойной точностью

- Fixxx память

Аналогично предыдущему типу команды, но операндами могут быть 16- или 32-разрядные целые числа

- Fxxx ST, ST(i)

Для этого типа регистр ST(i) является источником, а ST(0) - верхушка стека - приемником. Указатель стека не изменяется

- Fxxx ST(i), ST

Для этого типа регистр ST(0) является источником, а ST(i) - приемником. Указатель стека не изменяется

- FxxxP ST(i), ST

Регистр ST(i) - приемник, регистр ST(0) - источник. После выполнения команды источник ST(0) извлекается из стека

Строка "xxx" может принимать следующие значения:

- ADD - Сложение
- SUB - Вычитание
- SUBR - Обратное вычитание, уменьшаемое и вычитаемое меняются местами
- MUL - Умножение
- DIV - Деление
- DIVR - Обратное деление, делимое и делитель меняются местами

Кроме основных арифметических команд имеются дополнительные арифметические команды:

- FSQRT - Извлечение квадратного корня
- FSCALE - Масштабирование на степень числа 2
- FPREM - Вычисление частичного остатка
- FRNDINT - Округление до целого
- FXTRACT - Выделение порядка числа и мантиссы
- FABS - Вычисление абсолютной величины числа
- FCHS - Изменение знака числа

По команде FSQRT вычисленное значение квадратного корня записывается в верхушку стека ST(0).

Команда FSCALE изменяет порядок числа, находящегося в ST(0). По этой команде значение порядка числа ST(0) складывается с масштабным коэффициентом, который должен быть предварительно записан в ST(1). Действие этой команды можно представить следующей формулой:

ST(0) = ST(0) * 2n, где -215 <= n <= +215

В этой формуле n - это ST(1).

Команда FPREM вычисляет остаток от деления делимого ST(0) на делитель ST(1). Знак результата равен знаку ST(0), а сам результат получается в вершине стека ST(0).

Действие команды заключается в сдвигах и вычитаниях, аналогично ручному делению "в столбик". После выполнения команды флаг C2 регистра состояния может принимать следующие значения:

- 0 - Остаток от деления, полученный в ST(0), меньше делителя ST(1), команда завершилась полностью
- 1 - ST(0) содержит частичный остаток, программа должна еще раз выполнить команду для получения точного значения остатка

Команда RNDINT округляет ST(0) в соответствии с содержимым поля RC управляющего регистра.

Команда FABS вычисляет абсолютное значение ST(0). Аналогично, команда FCHS изменяет знак ST(0) на противоположный.

Трансцендентные команды

Трансцендентные команды предназначены для вычисления следующих функций:

- тригонометрические (sin, cos, tg,...)
- обратные тригонометрические (arcsin, arccos,...)
- показательные (xy, 2x, 10x, ex)
- гиперболические (sh, ch, th,...)
- обратные гиперболические (arsh, arch, arcth,...)

Вот список всех трансцендентных команд математического сопроцессора:

- FPTAN Вычисление частичного тангенса

- FPATAN Вычисление частичного арктангенса
- FYL2X Вычисление y*log2(x)
- FYL2XP1 Вычисление y*log2(x+1)
- F2XM1 Вычисление 2x-1
- FCOS Вычисление cos(x)
- FSIN Вычисление sin(x)
- FSINCOS Вычисление sin(x) и cos(x) одновременно

Команда FPTAN вычисляет частичный тангенс ST(0), размещая в стеке такие два числа x и y, что y/x = tg(ST(0)).

После выполнения команды число y располагается в ST(0), а число x включается в стек сверху (то есть записывается в ST(1)). Аргумент команды FPTAN должен находится в пределах:

0 <= ST(0) <= pi/4

Пользуясь полученным значением частичного тангенса, можно вычислить другие тригонометрические функции по следующим формулам:

- sin(z) = 2*(y/x) / (1 + (y/x)2)
- cos(z) = (1 - (y/x)2) / (1 + (y/x)2)
- tg(z/2) = y/x;
- ctg(z/2) = x/y;
- cosec(z) = (1 + (y/x)2) / 2*(y/x)
- sec(z) = (1 + (y/x)2) / (1 - (y/x)2)

Где z - значение, находившееся в ST(0) до выполнения команды FPTAN, x и y - значения в регистрах ST(0) и ST(1), соответственно.

Команда FPATAN вычисляет частичный арктангенс:

z=arctg(ST(0)/ST(1))=arctg(x/y).

Перед выполнением команды числа x и y располагаются в ST(0) и ST(1), соответственно. Аргументы команды FPATAN должен находится в пределах:

0 < y < x

Результат записывается в ST(0).

Команда FYL2X вычисляет выражение y*log2(x), операнды x и y размещаются, соответственно, в ST(0) и ST(1). Операнды извлекаются из стека, а результат записывается в стек. параметр x должен быть положительным числом.

Пользуясь результатом выполнения этой команды, можно вычислить следующим образом логарифмические функции:

- Логарифм по основанию два: log2(x) = FYL2(x)
- Натуральный логарифм: loge(x) = loge(2) * log2(x) = FYL2X(loge(2), x) = FYL2X(FLDLN2, x)
- Десятичный логарифм: log10(x) = log10(2) * log2(x) = FYL2X (log10(2), x) = FYL2X(FLDLG2, x)

Функция FYL2XP1 вычисляет выражение y*log2(x+1), где x соответствует ST(0), а y - ST(1). Результат записывается в ST(0), оба операнда выталкиваются из стека и теряются.

На операнд x накладывается ограничение: 0 < x < 1 - 1/sqrt(2)

Команда F2XM1 вычисляет выражение 2x-1, где x - ST(0). Результат записывается в ST(0), параметр должен находиться в следующих пределах: 0 <= x <= 0,5

Команда FCOS вычисляет cos(x). Параметр x должен находиться в ST(0), туда же записывается результат выполнения команды.

Команда FSIN аналогична команде FCOS, но вычисляет значение синуса ST(0).

Команда FSINCOS вычисляет одновременно значения синуса и косинуса параметра ST(0). Значение синуса записывается в ST(1), косинуса - в ST(0).

Пример ввода вещественного числа в математический сопроцессор (числа не более 1000, количество знаков после запятой не больше 4)

5.3 Пример

```
.286
.model small
.data
 buf_st db 10,0
```

```
 buf db 10 dup (0)
 c dw 0 ;целая часть числа
 d dw 0 ;дробная часть числа
 p dw 1 ;порядок числа
 res dd ?

.stack 128
.code
 .startup
 mov ah, 0ah
 lea dx, buf_st
 int 21h
 lea si,[buf]
l_celoe:
 cmp byte ptr [si], 13
 je l_end
 cmp byte ptr [si], '.'
 je l_drob

 mov bl, [si]
 sub bl, 30h
 mov bh, 0
 mov cx, 10
 mov ax, c
 mul cx
 add ax, bx
 mov c, ax
 inc si
 jmp l_celoe
l_drob:
 inc si
 cmp byte ptr [si], 13
 je l_end
 mov bl, [si]
 sub bl, 30h
 mov bh, 0
 mov cx, 10
 mov ax, d
 mul cx
 add ax, bx
 mov d, ax
 mov ax, p
 mul cx
 mov p, ax
 jmp l_drob
```

```
l_end:
 finit
 fild d
 fidiv p
 fiadd c
 fst res
 fwait

 mov ax, 4c00h
 int 21h
end
```

5.4 Задания

Рассчитать и вывести значение выражения, при заданных пользователем значениях x и a.

1) $$y = \frac{-15/6 + (34 - x*2)*2}{16 + \sin(a)/8} - 14$$

2) $$y = \frac{\frac{37 + 14*5}{\cos(4)} + 5*x*6}{17 - 4/2} + 5*4*e^{a}$$

3) $$y = 17*4 + \frac{5*a - 8/3}{\frac{14*\sin(x) + 5}{4} + \frac{16*6 + 5}{10}}$$

4) $$y = 35*4 + \frac{\frac{17 - 8/(3 + \cos(x))}{2} + 4*8}{3 + 2*e^{a}}$$

5) $$y = 3*6 + 4*4/(\sin(a)*3)*(48 + 16/2)*\frac{5 + 8*\cos(x)}{14 - 3}$$

6) $$y = \frac{14 - 4}{2}*(5*\sin(x - a) + 16/4)*\frac{17 + 8/3}{\frac{8}{4}*(1 + e^{2}) + 2*5}$$

7) $$y = \frac{171 + 4*3*2}{\frac{36*(\sin(x - e^{a})) - 4}{3} - 17} + (5*x + 6)*8*\frac{17}{5 + 2 - a}$$

8) $$y = \frac{250 - 4*6}{\frac{33*\cos(x + a) + 4}{2} - 17}*(2 + 4 - \sin(a)*e^{x}) - 17$$

9) $$y = 7 + 4*6 + 5/(2 - \sin(x + \frac{a}{3}))*13 + \frac{\frac{14 - 4}{2}*e^{x - a} + 17*6}{\frac{36 - 4}{3} + 2}$$

10) $$y=\frac{3*4+5/7}{5-(3-4*\sin(\frac{x}{5}+a))/2}-\frac{17*\cos(a)+2}{3*6+6*\ln(2)}$$

11) $$y=\frac{25/4+5*6}{36-e^{x*a}}+\frac{37*\sin(a+x)}{\frac{8*6+5}{3}+3*2}$$

12) $$y=\frac{37/2+\frac{4*6+5*e^{x-a}}{4}}{15-7}*(5+8*\frac{\sin(x)}{\cos(a)})-136$$

13) $$y=\frac{3*4+5*6-e^{x}*a}{7/2+4/3+12/3}*\frac{16}{2}3*6*\cos(\frac{a}{x})$$

14) $$y=\frac{49/7+42/6+6*2}{8*5/9-4*6}*(\cos(x)-\sin(a))+\frac{23*e^{x/a}+12}{26-4}*5$$

15) $$y=\frac{\frac{36+4*5}{2+6*\cos(x-a)}-14/7+1}{\frac{36-4}{2}+12*\sin(x+a)}-7*6*\frac{3*e^{x*a}+4}{12-7}$$

16) $$y=\frac{\frac{2+3}{2}*\cos(x)+\frac{4*6+25*5}{36-4}}{\frac{250*2-14}{37}*\sin(x-a)}+(12-4)*(37-8)*\ln(x)$$

17) $$y=\frac{\frac{320}{4}+\frac{256}{2}-14*\frac{\sin(x-a)}{\cos(x+a)}}{56*2*\lg(x-a)}*(3+2)$$

18) $$y=\frac{(12+4-7)*5}{36*4-\frac{17}{2}*e^{x-a}}*\frac{(2+5)}{7}*\cos(x+a)+14*\ln(a)$$

19) $$y=\frac{34+260/2+170*3*e^{x*a}}{\frac{36+8}{2}*\ln(x+a)+5*6}-\frac{34}{2}*(6*\cos(x*a)+7)$$

6 ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА ЗАДАЧ С ИСПОЛЬЗОВАНИЕМ СИСТЕМНЫХ РЕСУРСОВ BIOS. РАБОТА В ГРАФИЧЕСКОМ РЕЖИМЕ

6.1 Графический режим

Для генерации цветных изображений в графическом режиме используются минимальные точки растра - пиксели или пэлы (pixel).

При среднем разрешении каждый байт представляет четыре точки, пронумерованных от 0 до 3:

Байт: |C1 C0|C1 C0|C1 C0|C1 C0|

Пиксели: 0 1 2 3

В любой момент для каждой точки возможны четыре цвета, от 0 до 3. Ограничение в 4 цвета объясняется тем, что двухбитовая точка имеет 4 комбинации значений битов: 00, 01, 10 и 11. Можно выбрать значение 00 для любого из 16 возможных цветов фона или выбрать значение 01, 10, и 11 для одной из двух палитр. Каждая палитра имеет три цвета:

C1 C0 Палитра 0 Палитра 1

0 0 фон фон

0 1 зеленый голубой

1 0 красный сиреневый

1 1 коричневый белый

Для выбора цвета палитры и фона используется INT 10H. Таким образом, если, например, выбран фон желтого цвета и палитра 0, то возможны следующие цвета точки: желтый, зеленый, красный и коричневый. Байт, содержащий значение 10101010, соответствует красным точкам. Если выбрать цвет фона - синий и палитру 1, то возможные цвета: синий, голубой, сиреневый и белый. Байт, содержащий значение 00011011, отображает синюю, голубую, сиреневую и белую точки.

6.2 Прерывание BIOS INT 10H для графики

Функция AH=00 команды INT 10H устанавливает графический режим. Функция AH=11 команды INT 10H позволяет выбрать цвет палитры и вывести на экран графический символ. Код в регистре AH определяет функцию:

AH=00: Установка режима. Нулевое значение в регистре AH и 04 в регистре AL устанавливают стандартный цветной графический режим:

```
MOV AH,00 ;Функция установки режима
MOV AL,04 ;Разрешение 320x200
INT 10H
```

Установка графического режима приводит к исчезновению курсора с экрана. Подробности по установке режима приведены в главе 9.

AH=0BH: Установка цветовой палитры. Число в регистре BH определяет назначение регистра BL:

BH=00 выбирает цвета фона и бордюра в соответствии с содержимым регистра BL. Цвет фона от 1 до 16 соответствует шест. значениям от 0 до F; BH=01 выбирает палитру соответственно содержимому регистра BL (0 или 1):

```
MOV AH,0BH ;Функция установки цвета
MOV BH,01 ;Выбор палитры
MOV BL,00 ; 0 (зеленый, красный, корич.)
INT 10H ;Вызвать BIOS
```

Палитра, установленная один раз, сохраняется, пока не будет отменена другой командой. При смене палитры весь экран меняет цветовую комбинацию. При использовании функции AH=0BH в текстовом режиме, значение, установленное для цвета 0 в палитре, определяет цвет бордюра.

AH=0CH: Вывод точки на экран. Использование кода 0C в регистре AH позволяет вывести на экран точку в выбранном цвете (фон и палитра). Например, для разрешения 320x200 загрузим в регистр DX вертикальную координату (от 0 до 199), а в регистр CX - горизонтальную координату (от 0 до 319). В регистр AL поместим цвет точки (от 0 до 3):

MOV AH,0CH ;Функция вывода точки

MOV AL,цвет ;Цвет точки

MOV CX,столбец ;Горизонтальная координата

MOV DX,строка ;Вертикальная координата

INT 10H ;Вызвать BIOS

AH=0DH: Чтение точки с экрана. Данная функция позволяет прочитать точку для определения ее цвета. В регистр DX должна быть загружена вертикальная координата (от 0 до 199), а в регистр CX - горизонтальная (от 0 до 319). В регистре AH должно быть значение 0D. Функция возвращает цвет точки в регистре AL.

6.3 Задания

Построить и визуализировать график функции согласно варианту. Для расчета значения функции использовать математический сопроцессор. Должны быть также визуализированы оси координат.

Варианты заданий:

1) $y_x := \frac{13\cdot\cos(x) + 12\cdot\sin(x)}{e^x} - \sqrt{x^3 - x^2 + 45}$

2) $y_x := \sqrt{x^2 + 12\cdot x - 45} - x\cdot e^x$

3) $y_x := (\cos(x) - \sin(x))^2 - (\cos(x) + \sin(x))^3$

4) $y_x := \left|\frac{1}{x^2 + 1} - \ln(x)\right| \cdot \left(e^x \cdot \cos(x)\right)$

5) $y_x := \frac{\cos(x)^2 - \sin(x)^2}{\sin(x) + \cos(x)} \cdot e^{\frac{1}{x}}$

6) $y_x := \left(15\cdot x^3 + 12\cdot x^2 + 17\cdot x + 34\right) \cdot \frac{1}{x^2 - 23}$

7) $y_x := \frac{\ln\left(x + \sqrt{x^2 - 18\cdot x}\right)}{\log\left(x + e^{x^2 - 12}\right)}$

8) $y_x := \left(\cos\left(x^2 + 23\cdot x - 76\right) + \sin\left(x^3 - 12\cdot x + 49\right)\right) \cdot e^{x-10}$

9) $y_x := \sin(\pi - x) \cdot \ln\left(x^2 + 8 \cdot x\right) + \cos\left(\sqrt{x^2 - 12}\right) \cdot e^{-x}$

10) $y_x := \dfrac{x^4 - 12 \cdot x^3 + 31 \cdot x}{21 \cdot x^2 - 17} - \dfrac{15 \cdot x^3 + 14}{x + 5}$

11) $y_x := \ln\left(\dfrac{x^3 - 18}{x^2 + 11}\right) - e^{\sqrt{x^2+26}} \cdot \cos(\pi - x)$

12) $y_x := e^{\frac{x^4+13\cdot\sqrt{x-14}}{x^5-12\cdot x^2}} + 2x^{\frac{3}{x}}$

13) $y_x := \sin\left(\dfrac{x^2 - 17}{x^3 + 15}\right) + \dfrac{\cos\left(\dfrac{x^3 + 22}{x + 16}\right)}{\sin\left(e^x\right)}$

14) $y_x := \left(\sin\left(x - \dfrac{\pi}{2}\right) + \cos\left(10 \cdot x - x^2\right)\right)^2 - \ln\left(x^3 - \sqrt{3x - 12}\right)$

15) $y_x := \cos\left(\dfrac{\pi}{3} - x\right) \cdot \dfrac{5 \cdot x^3 - 22}{17 \cdot x^2 - 5} + e^{\frac{-1}{x}}$

16) $y_x := \left(5 \cdot x^4 - 7 \cdot x^3\right) \cdot \log\left(e^{-x}\right) + 12 + \cos\left(x^2 - 17\right)$

17) $y_x := \left(3 \cdot x^3 - 12 \cdot x^2 + 15\right) \cdot \left(\dfrac{\cos\left(x^2 - 5 \cdot x\right)}{\sin\left(\dfrac{\pi}{3} \cdot 5 - x\right)}\right)$

18) $y_x := \dfrac{-12 \cdot x^4 + 5 \cdot x^3 - 23}{e^{13x - x^2}} \cdot \cos\left(\log\left(x^2\right) - 15 \cdot \dfrac{\pi}{7}\right)$

7 РАБОТА С ФАЙЛАМИ В ЯЗЫКЕ ASSEMBLER

7.1 Создание файла

Функция DOS 3Ch — Создать файл

Ввод:	AX = 3Ch CX = атрибут файла бит 7: файл можно открывать разным процессам в Novell Netware бит 6: не используется бит 5: архивный бит (1, если файл не сохранялся) бит 4: каталог (должен быть 0 для функции 3Ch) бит 3: метка тома (игнорируется функцией 3Ch) бит 2: системный файл бит 1: скрытый файл бит 0: файл только для чтения DS:DX = адрес ASCIZ-строки с полным именем файла (ASCIZ-строка ASCII-символов, оканчивающаяся нулем)
Вывод:	CF = 0 и AX = идентификатор файла, если не произошла ошибка CF = 1 и AX = 03h, если путь не найден CF = 1 и AX = 04h, если слишком много открытых файлов CF = 1 и AX = 05h, если доступ запрещен

Если файл уже существует, функция 3Ch все равно открывает его, присваивая ему нулевую длину. Чтобы этого не произошло, следует пользоваться функцией 5Bh.

Пример:

```
mov al,00h
lea dx,name1 ; устанавливаем имя файла name1
mov ah,3ch
int 21h ; создаем файл
```

7.2 Открытие существующего файла

Ввод:	AX = 3Dh AL = режим доступа биты 0 – : права доступа 00: чтение 01: запись 10: чтение и запись бит 1: открыть для записи биты 2 – 3: зарезервированы (0) биты 6 – 4: режим доступа для других процессов 000: режим совместимости (остальные процессы также должны открывать этот файл в режиме совместимости) 001: все операции запрещены 010: запись запрещена 011: чтение запрещено 100: запрещений нет бит 7: файл не наследуется порождаемыми процессами DS:DX = адрес ASCIZ-строки с полным именем файла CL = маска атрибутов файлов
Вывод:	CF = 0 и AX = идентификатор файла, если не произошла ошибка CF = 1 и AX = код ошибки (02h — файл не найден, 03h — путь не найден, 04h — слишком много открытых файлов, 05h — доступ запрещен, 0Ch — неправильный режим доступа)

Пример:

```
mov al,02h ; открываем файл на чтение\запись
lea dx,name1 ; устанавливаем имя файла name1
mov ah,3dh
int 21h ; открываем файл
```

7.3 Создание и открытие файла.

Создать и открыть новый файл

Ввод:	AX = 5Bh CX = атрибут файла DS:DX = адрес ASCIZ-строки с полным именем файла
Вывод:	CF = 0 и AX = идентификатор файла, открытого для чтения/записи в режиме совместимости, если не произошла ошибка CF = 1 и AX = код ошибки (03h — путь не найден, 04h — слишком много открытых файлов, 05h — доступ запрещен, 50h — файл уже существует)

Функция DOS 5Ah — Создать и открыть временный файл

Ввод:	AX = 5Ah CX = атрибут файла DS:DX = адрес ASCIZ-строки с путем, оканчивающимся символом «\», и тринадцатью нулевыми байтами в конце
Вывод:	CF = 0 и AX = идентификатор файла, открытого для чтения/записи в режиме совместимости, если не произошла ошибка (в строку по адресу DS:DX дописывается имя файла) CF = 1 и AX = код ошибки (03h — путь не найден, 04h — слишком много открытых файлов, 05h — доступ запрещен)

Функция 5Ah создает файл с уникальным именем, который не является на самом деле временным, его следует специально удалять, для чего его имя и записывается в строку в DS:DX.

Во всех случаях строка с полным именем файла имеет вид типа

```
Name1 db 'c:\data\filename.ext',0
```

причем, если диск или путь опущены, используются их текущие значения.

Для работы с длинными именами файлов в DOS 7.0 (Windows 95) и старше используется еще один дополнительный набор функций, которые вызываются как функция DOS 71h.

Функция LFN 6Ch — Создать или открыть файл с длинным именем

Ввод:	AX = 716Ch BX = режим доступа Windows 95 биты 2 – 0: доступ 000 — только для чтения 001 — только для записи 010 — для чтения и записи 100 — только для чтения, не изменять время последнего обращения к файлу биты 6 – 4: доступ для других процессов (см. функцию 3Dh) бит 7: файл не наследуется порождаемыми процессами бит 8: данные не буферизуются бит 9: не архивировать файл, если используется архивирование файловой системы (DoubleSpace) бит 10: использовать число в DI для записи в конец короткого имени файла бит 13: не вызывать прерывание 24h при критических ошибках бит 14: сбрасывать буфера на диск после каждой записи в файл CX = атрибут файла DX = действие бит 0: открыть файл (ошибка, если файл не существует) бит 1: заменить файл (ошибка, если файл не существует) бит 4: создать файл (ошибка, если файл существует) DS:SI = адрес ASCIZ-строки с именем файла DI = число, которое будет записано в конце короткого варианта имени файла
Вывод:	CF = 0 AX = идентификатор файла CX = 1, если файл открыт CX = 2, если файл создан CX = 3, если файл заменен CF = 1, если произошла ошибка AX = код ошибки (7100h, если функция не поддерживается)

Если функции открытия файлов возвращают ошибку «слишком много открытых файлов» (AX = 4), следует увеличить число допустимых идентификаторов с помощью функции 67h.

7.4 Чтение, запись и переименование файла

Чтение из файла или устройства

Ввод:	AH = 3Fh BX = идентификатор CX = число байт DS:DX = адрес буфера для приема данных
Вывод:	CF = 0 и AX = число считанных байт, если не произошла ошибка CF = 1 и AX = 05h, если доступ запрещен, 06h, если неправильный идентификатор

Пример:

```
  mov bx,ax ; идентификатор файла в BX
mov cx,1 ; считывать один байт
mov dx,offset buffer ; начало буфера - в DX
mov ah,3Fh ; чтение файла
int 21h
```

Если при чтении из файла число фактически считанных байт в AX меньше, чем заказанное число в CX, при чтении был достигнут конец файла. Каждая следующая операция чтения, так же как и записи, начинается не с начала файла, а с того байта, на котором остановилась предыдущая операция чтения/записи. Если требуется считать (или записать) произвольный участок файла, используют функцию 42h (функция lseek в C).

7.5 Перемещение указателя чтения/записи

Ввод:	AH = 42h BX = идентификатор CX:DX = расстояние, на которое надо переместить указатель (со знаком) AL = перемещение от: 0 — от начала файла 1 — от текущей позиции 2 — от конца файла
Вывод:	CF = 0 и CX:DX = новое значение указателя (в байтах от начала файла), если не произошла ошибка CF = 1 и AX = 06h, если неправильный идентификатор

Пример:

```
mov ax,4201h ; переместить указатель файла от текущей
dec cx ; позиции назад на 1
dec cx ; CX = FFFFh
mov dx,cx ; DX = FFFFh
int 21h
```

Указатель можно установить за реальными пределами файла: если указатель устанавливается в отрицательное число, следующая операция чтения/записи вызовет ошибку; если указатель устанавливается в положительное число, большее длины файла, следующая операция записи увеличит размер файла. Эта функция также часто используется для определения длины файла — достаточно вызвать ее с CX = 0, DX = 0, AL = 2, и в CX:DX будет возвращена длина файла в байтах.

7.6 Запись в файл или устройство

Ввод:	AH = 40h BX = идентификатор CX = число байт DS:DX = адрес буфера с данными
Вывод:	CF = 0 и AX = число записанных байт, если не произошла ошибка CF = 1 и AX = 05h, если доступ запрещен, 06h, если неправильный идентификатор

Если при записи в файл указать CX = 0, файл будет обрезан по текущему значению указателя. При записи в файл на самом деле происходит запись в буфер DOS, данные из которого сбрасываются на диск при закрытии файла или если их количество превышает размер сектора диска. Для немедленного сброса буфера можно использовать функцию 68h (функция fflush в C).

Пример: с переносом указателя и записью в файл

```
mov ax,4201h ; переместить указатель файла от текущей
dec cx ; позиции назад на 1
dec cx ; CX = FFFFh
mov dx,cx ; DX = FFFFh
int 21h
mov ah,40h ; записать в файл
```

7.7 Переименование файла

Ввод:	AH = 56h DS:DX = ASCIIZ- имя существующего файла ES:DI = ASCIIZ- имя нового файла CL = Маска атрибутов
Вывод:	CF = 0, если операция выполнена CF = 1, если произошла ошибка (AX = код ошибки)

Функция 56h позволяет произвести перемещение между каталогами, не изменяя устройства.

Пример: Перемещения между каталогами без смены устройства

```
.data
fname_s db ‘name1.asm’
point_fname_s dd fname_s
fname_d db ‘e:\name1.asm’
point_fname_d dd fname_d
.
.
.
        .code ; переместим файл из текущего каталога в корневой
         lds dx.point_fname_s ; указатель на исходный файл
les di.point_fname_d ; указатель на новый файл
mov ah,56h
int 21h ; перемещаем файл
```

7.8 Закрытие и удаление файла

7.8.1 Закрыть файл

Ввод:	AH = 3Eh BX = идентификатор
Вывод:	CF = 0, если не произошла ошибка CF = 1 и AX = 6, если неправильный идентификатор

Если файл был открыт для записи, все файловые буфера сбрасываются на диск, устанавливается время модификации файла и записывается его новая длина.

7.8.2 Удаление

Функция DOS 41h — Удаление файла

Ввод:	AH = 41h DS:DX = адрес ASCIZ-строки с полным именем файла
Вывод:	CF = 0, если файл удален CF = 1 и AH = 02h, если файл не найден, 03h — если путь не найден, 05h — если доступ запрещен

Удалить файл можно только после того, как он будет закрыт, так как DOS будет продолжать выполнять запись в несуществующий файл, что может привести к разрушению файловой системы. Функция 41h не позволяет использовать маски (символы * и ? в имени файла) для удаления сразу нескольких файлов, хотя этого можно добиться, вызывая ее через недокументированную функцию 5D00h. Но, начиная с DOS 7.0 (Windows 95), официальная функция удаления файла может работать сразу с несколькими файлами.

7.9 Удаление файлов с длинным именем

Ввод:	AX = 7141h DS:DX = адрес ASCIZ-строки с длинным именем файла SI = 0000h: маски не разрешены и атрибуты в CX игнорируются SI = 0001h: маски в имени файла и атрибуты в CX разрешены: CL = атрибуты, которые файлы могут иметь CH = атрибуты, которые файлы должны иметь
Вывод:	CF = 0, если файл или файлы удалены CF = 1 и AX = код ошибки, если произошла ошибка. Код 7100h означает, что функция не поддерживается

7.10 Поиск файлов

7.10.1 Найти первый файл

Найти нужный файл на диске намного сложнее, чем просто открыть его, — для этого требуются две функции при работе с короткими именами (найти первый файл и найти следующий файл) и три — при работе с длинными именами в DOS 7.0 (найти первый файл, найти следующий файл, прекратить поиск).

Функция DOS 4Eh — Найти первый файл

Ввод:	AH = 4Eh AL используется при обращении к функции APPEND CX = атрибуты, которые должен иметь файл (биты 0 (только для чтения) и 5 (архивный бит) игнорируются, если бит 3 (метка тома) установлен, все остальные биты игнорируются) DS:DX = адрес ASCIZ-строки с именем файла, которое может включать путь и маски для поиска (символы * и ?)
Вывод:	CF = 0 и область DTA заполняется данными, если файл найден CF = 1 и AX = 02h, если файл не найден, 03h — если путь не найден, 12h — если неправильный режим доступа

Вызов этой функции заполняет данными область памяти DTA (область передачи данных), которая начинается по умолчанию со смещения 0080h от начала блока данных PSP (при запуске COM- и EXE-программ сегменты DS и ES содержат

сегментный адрес начала PSP), но ее можно переопределить с помощью функции 1Ah.

Функция DOS 1Ah — Установить область DTA

Ввод:	AH = 1Ah DS:DX = адрес начала DTA (128-байтный буфер)

Функции поиска файлов заполняют DTA следующим образом:

+00h: байт — биты 0 – 6: ASCII-код буквы диска; бит 7: диск сетевой

+01h: 11 байт — маска поиска (без пути)

+0CH: байт — атрибуты для поиска

+0Dh: слово — порядковый номер файла в каталоге

+0Fh: слово — номер кластера начала внешнего каталога

+11h: 4 байта — зарезервировано

+15h: байт — атрибут найденного файла

+16h: слово — время создания файла в формате DOS:

биты 15 – 11: час (0 — 23)

биты 10 – 5: минута

биты 4 – 0: номер секунды, деленный на 2 (0 – 30)

+18h: слово — дата создания файла в формате DOS:

биты 15 – 9: год, начиная с 1980

биты 8 – 5: месяц

биты 4 – 0: день

+1Ah: 4 байта — размер файла

+1Eh: 13 байт — ASCIZ-имя найденного файла с расширением

После того как DTA заполнена данными, для продолжения поиска следует вызывать функцию 4Fh, пока не будет возвращена ошибка.

7.10.2 Найти следующий файл

Функция DOS 4Fh — Найти следующий файл

Ввод:	AH = 4Fh DTA — содержит данные от предыдущего вызова функции 4E или 4F
Вывод:	CF = 0 и DTA содержит данные о следующем найденном файле, если не произошла ошибка CF = 1 и AX = код ошибки, если произошла ошибка

Для случая длинных имен файлов (LFN) употребляется набор из трех подфункций функции DOS 71h, которые можно использовать, только если запущен IFSmgr (всегда запускается при обычной установке Windows 95, но не запускается, например, с загрузочной дискеты MS-DOS 7.0).

В качестве примера программы, использующей многие из функций работы с файлами, рассмотрим программу, заменяющую русские буквы «Н» на латинские «H» во всех файлах с расширением .TXT в текущем каталоге.

```
; fidoh.asm
; заменяет русские "Н" на латинские "H" во всех файлах с расширением .TXT
; в текущем каталоге
 .model tiny
 .code
 org 100h ; COM-файл
start:
 mov ah,4Eh ; поиск первого файла
 xor cx,cx ; не системный, не каталог и т.д.
 mov dx,offset filespec ; маска для поиска в DS:DX
file_open:
 int 21h
 jc no_more_files ; если CF = 1 - файлы кончились

 mov ax,3D02h ; открыть файл для чтения и записи
 mov dx,80h+1Eh ; смещение DTA + смещение имени файла
 int 21h ; от начала DTA
 jc find_next ; если файл не открылся - перейти
 ; к следующему
 mov bx,ax ; идентификатор файла в BX
 mov cx,1 ; считывать один байт
 mov dx,offset buffer ; начало буфера - в DX
read_next:
```

```
 mov ah,3Fh ; чтение файла
 int 21h
 jc find_next ; если ошибка - перейти к следующему
 dec ax ; если AX = 0 - файл кончился -
 js find_next ; перейти к следующему
 cmp byte ptr buffer,8Dh ; если не считана русская "Н",
 jne read_next ; считать следующий байт,
 mov byte ptr buffer,48h ; иначе - записать в буфер
 ; латинскую букву "H"
 mov ax,4201h ; переместить указатель файла от текущей
 dec cx ; позиции назад на 1
 dec cx ; CX = FFFFh
 mov dx,cx ; DX = FFFFh
 int 21h
 mov ah,40h ; записать в файл
 inc cx
 inc cx ; один байт (CX = 1)
 mov dx,offset buffer ; из буфера в DS:DX
 int 21h
 jmp short read_next ; считать следующий байт
find_next:
 mov ah,3Eh ; закрыть предыдущий файл
 int 21h
 mov ah,4Fh ; найти следующий файл
 mov dx,80h ; смещение DTA от начала PSP
 jmp short file_open
no_more_files: ; если файлы кончились,
 ret ; выйти из программы
filespec db "*.txt",0 ; маска для поиска
buffer label byte ; буфер для чтения/записи -
 end start ; за концом программы
```

7.11 Задания

Во всех вариантах необходимо реализовать программу работы с файлами. Пользователь вводит с клавиатуры имя файла с текстом и имя создаваемого файла, в который будет помещен результат. Слова в строке могут быть разделены пробелами и знаками препинания.

Варианты заданий:

1) Выровнять все строки в файле по правому краю
2) Выровнять все строки в файле по центру

3) Выровнять все строки в файле по левому краю
4) Отформатировать файл таким образом, чтобы длина каждой строки не превышала заданного пользователем значения
5) Отсортировать строки файла по длинам
6) Отсортировать строки файла по количеству гласных букв
7) Отсортировать строки файла по алфавиту по первым трем буквам
8) Отсортировать строки файла по количеству одинаковых букв
9) Найти в каждой строке файла заданную пользователем последовательность символов и заменить на введенное слово

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Абель, Питер Assembler и программирование для IBM PC [Электронный ресурс]. - http://www.proklondike.com/books/assembler/assembler_abel_ibmpc.html.
2) Юров, В. И. Assembler. Практикум [Электронный ресурс]. - http://www.proklondike.com/books/assembler/Urov_Assembler_praktikum_2006.html.
3) Юров, В. И. Assembler. Учебник для вузов [Электронный ресурс]. - http://www.proklondike.com/books/assembler/yurov_dlyavuzov.html.
4) Зубков, С.В. Assembler. Язык неограниченных возможностей [Электронный ресурс]. - http://www.proklondike.com/books/assembler/zubkov.html.
5) Пильщиков, В.Н. Программирование на языке ассемблера IBM PC [Текст]. – СПб.: Изд-во Питре, 2003. – 258 с.

СПИСОК ВОПРОСОВ К ЗАЧЕТУ
по дисциплине «Организация ЭВМ и вычислительных систем**»**
для направления 010300.2.62
«Фундаментальные информатика и информационные технологии»

1. Основные характеристики ЭВМ.
2. Классификация средств ЭВМ.
3. Общие принципы построения современных ЭВМ.
4. История развития ВТ (ЭВМ) 1-3 поколение ЭВМ.
5. История развития ВТ (ЭВМ) 3-5 поколение ЭВМ.
6. Модульность построения, магистральность, иерархия управления.
7. Иерархический принцип построения памяти.
8. Классификация программного обеспечения.
9. Показатели качества программного продукта (ПП).
10. Системы счисления. Представление числовой информации в ЭВМ.
11. Арифметические основы ЭВМ.
12. Машинные коды. Классификация, принципы кодирования информации.
13. Выполнение операций сложения (вычитание) в машинных кодах.
14. Выполнение операций умножение (деление) в машинных кодах.
15. Арифметические операции над двоичными числами с плавающей точкой.
16. Арифметические операции над двоично-десятичными кодами чисел.
17. Основные сведения из алгебры логики.
18. Законы алгебры логики.
19. Понятие о минимизации логических функций .Техническая интерпретация логических функций
20. Классификация элементов и узлов ЭВМ
21. Комбинационные схемы. Дешифратор.
22. Комбинационные схемы. Шифратор.
23. Комбинационные схемы. Компаратор.
24. Схемы с памятью. Триггеры. Классификация триггеров.
25. Схемы с памятью. Одноступенчатые и двухступенчатые RS-триггеры.
26. Схемы с памятью. Т-триггеры, JK-триггеры, D-триггеры.
27. Узлы ЭВМ. Регистры хранения.
28. Узлы ЭВМ. Регистр сдвига.
29. Узлы ЭВМ. Счетчик.
30. Узлы ЭВМ. Сумматор.
31. Узлы ЭВМ. Арифметико-логическое устройство (АЛУ).
32. Узлы ЭВМ. Устройство управления (УУ).
33. Датчик сигналов на основе счетчика с дешифратором и на сдвиговом регистре.
34. Структурная схема микропрограммного устройства управления.
35. Проблемы развития элементной базы.
36. Общие принципы функциональной и структурной организации ЭВМ.
37. Организация функционирования ЭВМ с магистральной архитектурой.
38. Организация работы ЭВМ при выполнении задания пользователя.
39. Отображение адресного пространства программы на основную память.
40. Адресная структура команд микропроцессора и планирование ресурсов.
41. Организация виртуальной памяти.
42. Система прерываний ЭВМ.

Доцент кафедры ПС,к.т.н. Зеленко Л.С.

Список вопросов к проведению теста по дисциплине «Организация ЭВМ и вычислительных систем»

1. **Компьютер - это универсальное устройство для:**

- ☐ ввода информации
- ☐ проведения вычислений
- ☑ обработка информации

2. **Что такое компьютер?**

- ☑ это устройство или система, способная выполнять заданную, четко отработанную последовательность операций по обработке информации
- ☐ это устройство или система, способная выполнить любую задачу, поставленную пред пользователем
- ☐ это устройство или система не способная выполнять заданную, четко отработанную последовательность операций по обработке информации

3. **Что такое программное обеспечение?**

- ☑ это совокупность программ, позволяющая организовывать решение задач пользователя на компьютере
- ☐ это совокупность аппаратных устройств, позволяющих организовать решение задач пользователя
- ☐ это математические методы, позволяющие решить задачу пользователя на компьютере.

4. **Системные программы служат для:**

- ☐ решения математических задач
- ☐ считывания данных с компьютера
- ☑ управление ресурсами компьютера

5. **В устройстве IBM-совместимых компьютеров используется принцип:**

- ☐ суперпозиции
- ☐ интеграции
- ☑ открытой архитектуры

6. **Из перечисленных принципов указать не составляющий принцип открытой архитектуры:**

- ☐ компьютер можно собирать из отдельных комплектующих, разработанных и изготовленных независимыми фирмами - изготовителями
- ☑ современные компьютеры построены на принципах программного управления, односторонности памяти и адресности
- ☐ компьютер легко модернизируется за счет наличия внутренних расширительных гнезд для разнообразных устройств

7. **За что отвечает микропроцессор?**

- ☑ за быстродействие
- ☐ за качество видеоизображения
- ☐ за сохранение информации

8. **На скорость работы компьютера влияет:**

- ☐ объем жесткого диска
- ☐ тип монитора
- ☑ тип процессора

9. **Какая компания является ведущей в производстве микропроцессоров:**

- ☐ MICROSOFT
- ☑ INTEL
- ☐ EPSON

10. **Для чего предназначен жесткий диск (винчестер):**

- ☐ для временного хранения информации
- ☐ для длительного хранения и считывания информации, которая изменяется редко
- ☑ для любого вида хранения информации (временного и длительного) и считывания информации

11. **Винчестер - это:**

- ☐ устройство оптимизации работы компьютера
- ☑ жесткий диск
- ☐ поддержка работы процессора

12. **Для связи компьютера с компьютером используется устройство:**

- ☑ шина данных
- ☐ модем
- ☐ сканер

13. **Сканер это устройство для:**

- ☑ обработки информации
- ☐ вывода информации
- ☐ ввода графической и текстовой информации

14. **Из перечисленных устройств выберите основное устройство:**

- ☐ модем
- ☐ сканер
- ☑ системный блок (процессор)

15. **Компьютер обрабатывает информацию в ____ системе счисления:**

- ☑ двоичной
- ☐ шестнадцатеричной
- ☐ десятичной

Компьютер – это:

- устройство для работы с текстами;
- электронное вычислительное устройство для обработки чисел;
- устройство для хранения информации любого вида;
- многофункциональное электронное устройство для работы с информацией;
- устройство для обработки аналоговых сигналов.

2. Какое устройство в компьютере служит для обработки информации?

- манипулятор "мышь"
- процессор
- клавиатура
- монитор
- оперативная память

3. Скорость работы компьютера зависит от:

- тактовой частоты обработки информации в процессоре;
- наличия или отсутствия подключенного принтера;
- организации интерфейса операционной системы;
- объема внешнего запоминающего устройства;
- объема обрабатываемой информации.

4. Тактовая частота процессора – это:

- число двоичных операций, совершаемых процессором в единицу времени;
- число вырабатываемых за одну секунду импульсов, синхронизирующих работу узлов компьютера;
- число возможных обращений процессора к операционной памяти в единицу времени;
- скорость обмена информацией между процессором и устройствами ввода/вывода;
- скорость обмена информацией между процессором и ПЗУ.

5. Объем оперативной памяти определяет:

- какой объем информации может храниться на жестком диске
- какой объем информации может обрабатываться без обращений к жесткому диску

- какой объем информации можно вывести на печать
- какой объем информации можно копировать

6. Укажите наиболее полный перечень основных устройств:

- микропроцессор, сопроцессор, монитор;
- центральный процессор, оперативная память, устройства ввода/вывода;
- монитор, винчестер, принтер;
- АЛУ, УУ, сопроцессор;
- сканер, мышь, монитор, принтер.

7. Магистрально-модульный принцип архитектуры современных персональных компьютеров подразумевает такую логическую организацию его аппаратных компонентов, при которой:

- каждое устройство связывается с другими напрямую;
- каждое устройство связывается с другими напрямую, а также через одну центральную магистраль;
- все они связываются друг с другом через магистраль, включающую в себя шины данных, адреса и управления;
- устройства связываются друг с другом в определенной фиксированной последовательности (кольцом);
- связь устройств друг с другом осуществляется через центральный процессор, к которому они все подключаются.

8. Назовите устройства, входящие в состав процессора:

- оперативное запоминающее устройство, принтер;
- арифметико-логическое устройство, устройство управления;
- кэш-память, видеопамять;
- сканер, ПЗУ;
- дисплейный процессор, видеоадаптер.

9. Процессор обрабатывает информацию:

- в десятичной системе счисления
- в двоичном коде
- на языке Бейсик
- в текстовом виде

10. Постоянное запоминающее устройство служит для:

- сохранения программ начальной загрузки компьютера и тестирования его узлов;
- хранения программы пользователя во время работы;
- записи особо ценных прикладных программ;

- хранения постоянно используемых программ;
- постоянного хранения особо ценных документов.

11. Во время исполнения прикладная программа хранится:

- в видеопамяти;
- в процессоре;
- в оперативной памяти;
- на жестком диске;
- в ПЗУ.

12. Адресуемость оперативной памяти означает:

- дискретность структурных единиц памяти;
- энергозависимость оперативной памяти;
- возможность произвольного доступа к каждой единице памяти;
- наличие номера у каждой ячейки оперативной памяти;
- энергонезависимость оперативной памяти.

13. Персональный компьютер не будет функционировать, если отключить:

- дисковод;
- оперативную память;
- мышь;
- принтер;
- сканер.

14. Для долговременного хранения информации служит:

- оперативная память;
- процессор;
- внешний носитель;
- дисковод;
- блок питания.

15. Процесс хранения информации на внешних носителях принципиально отличается от процесса хранения информации в оперативной памяти:

- тем, что на внешних носителях информация может храниться после отключения питания компьютера;
- объемом хранимой информации;
- различной скоростью доступа к хранимой информации;

- возможностью защиты информации;
- способами доступа к хранимой информации.

16. При отключении компьютера информация:

- исчезает из оперативной памяти;
- исчезает из постоянного запоминающего устройства;
- стирается на «жестком диске»;
- стирается на магнитном диске;
- стирается на компакт-диске.

17. Дисковод – это устройство для:

- обработки команд исполняемой программы;
- чтения/записи данных с внешнего носителя;
- хранения команд исполняемой программы;
- долговременного хранения информации;
- вывода информации на бумагу.

18. Какое устройство обладает наибольшей скоростью обмена информацией?

- CD-ROM дисковод
- жесткий диск
- дисковод для гибких дисков
- микросхемы оперативной памяти

19. Какое из устройств предназначено для ввода информации:

- процессор;
- принтер;
- ПЗУ;
- клавиатура;
- монитор.

20. Манипулятор «мышь» – это устройство:

- модуляции и демодуляции;
- считывания информации;
- долговременного хранения информации;
- ввода информации;
- для подключения принтера к компьютеру.

21. Для подключения компьютера к телефонной сети используется:

- модем;

- факс;
- сканер;
- принтер;
- монитор.

22. Принцип программного управления работой компьютера предполагает:

- двоичное кодирование данных в компьютере;
- моделирование информационной деятельности человека при управлении компьютером;
- необходимость использования операционной системы для синхронной работы аппаратных средств;
- возможность выполнения без внешнего вмешательства целой серии команд;
- использование формул исчисления высказываний для реализации команд в компьютере.

23. Файл – это:

- именованный набор однотипных элементов данных, называемых записями;
- объект, характеризующийся именем, значением и типом;
- совокупность индексированных переменных;
- совокупность фактов и правил;
- терм.

24. Расширение имени файла, как правило, характеризует:

- время создания файла;
- объем файла;
- место, занимаемое файлом на диске;
- тип информации, содержащийся в файле;
- место создания файла

Вычислительные системы по принципу закрепления вычислительных функций делятся на:

- системы с плавающим закреплением функций
- однородные
- неоднородные
- системы с жестким закреплением функций

__ - система организующая, хранящая и преобразующая информацию (то есть система, основным предметом и продуктом труда которой является информация)

- Операционная система
- Драйвер
- Информационная система
- Утилита

Важнейшими принципами построения АСОД являются принципы:

- комплексности
- интеграции
- совместимости
- системности

Вычислительные машины (ВМ) по назначению делятся на:

- специализированные
- универсальные
- проблемно-ориентированные
- гибридные

Вычислительные машины (ВМ) по принципу действия делятся на:

- аналоговые
- цифровые
- автоматические
- гибридные

Вычислительные системы по режиму работы делятся на:

- работающие в оперативном временном режиме
- однородные
- неоднородные
- работающие в неоперативном временном режиме

Вычислительные системы по типу ЭВМ и процессоров делятся на:

- системы с плавающим закреплением функций
- системы с жестким закреплением функций
- однородные
- неоднородные

Основными принципами построения ВС являются:

- унификация и стандартизация технических и программных решений
- модульность структуры технических и программных средств
- неспособность к адаптации
- работа в разных режимах

Установите связь между типом ЭВМ и ее функциями

универсальные	-	предназначены для решения более узкого круга задач, связанных, как правило, с управлением технологическими объектами
проблемно-ориентированные	-	предназначены для решения определенного, узкого круга задач или реализации строго определенной группы функций
специализированные	-	предназначены для решения самых различных задач, которые отличаются сложностью алгоритмов и большим объемом обрабатываемых данных

Что из перечисленного ниже определяется типом процессора?

- ☐ Производительность ПК
- ☐ Объем оперативной памяти

- Разрешающая способность монитора
- Тип чипсета материнской платы
- Набор внешнего оборудования

Для чего в компьютере используется оперативная память?

- Для постоянного хранения данных
- Для постоянного хранения команд
- Для временного, только во время работы компьютера хранения команд
- Для временного, только во время работы компьютера хранения данных
- Для постоянного хранения системы BIOS

Какие функции выполняет процессор в вычислительной системе?

- Функции управления вычислительной системой
- Вычислительные функции
- Процессор осуществляет ввод и вывод данных
- Процессор передает функции управления вводом и выводом данных специальному контроллеру ввода/вывода, который управляет процессором
- Процессор передает функции управления вводом и выводом данных специальному контроллеру ввода/вывода, и управляет этим контроллером
- Возможны ли прерывания работы центрального процессора для работы с устройствами ввода/вывода.
- Управление процессом ввода/вывода осуществляется программой ввода/вывода, которая считывается из основной памяти и выполняется процессором ввода/вывода автономно.
- Управление процессом ввода/вывода осуществляется программой ввода/вывода, которая считывается из основной памяти и выполняется центральным процессором, процессор ввода/вывода не требуется.
- Процессор читает из памяти программу, состоящую из команд, каждая команда выполняется как набор микроопераций
- Процессор читает из памяти программу, состоящую из микроопераций, каждая из которых выполняет элементарное действие

В чем заключается сущность микропрограммного управления

- Любая машинная операция выполняется при помощи специальной схемы управления.
- Любая машинная операция выполняется как последовательность микроопераций.
- Микропрограмма хранится в специальном запоминающем устройстве, как последовательность управляющих слов.
- Вместо аппаратного формирования управляющих сигналов используется микропрограмма.
- Сигналы управления последовательно читаются из памяти микропрограмм и устанавливаются на соответствующих линиях управления.
- Для передачи сигналов управления используется шина данных.
- Для передачи сигналов управления используется специальная шина - шина управления.

Для чего нужна дисковая память компьютеру?

- **Для долговременного хранения программ и данных**
- **Для хранения промежуточных данных выполняемых процессором операций**
- **Для хранения операционной системы**
- **Для хранения системы BIOS**
- **Для хранения системы Setap**
- **Для хранения системы POST**

Как организована сеть Internet?

- Имеется главный сервер, распределяющий потоки данных.
- Множество серверов в сети равноправно выполняют функции передачи и приема данных
- Как переданные данные находят своего адресата? Каждый сервер, отправляя пакет данных формирует адрес приемника и пакет движется по сетевым соединениям, минуя все другие серверы, пока не найдет своего адресата
- Выход в Internet обеспечивает провайдер
- Любой компьютер может быть узлом сети Internet
- Каждый узел имеет свой уникальный адрес
- Каждая связь с удаленным компьютером осуществляется в сети Internet по одному, раз установленному пути

- Каждое сообщение разбивается на пакеты, пакеты могут распространяться по разным путям
- Узел просматривает все пакеты и пересылает дальше не адресованные ему пакеты
- Узел просматривает только адресованные ему пакеты

Что Вы можете сказать о компьютерах пятого поколения?

- Для перехода к ЭВМ 5-го поколения было определено требование перехода к новой технологической базе.
- Современные компьютеры можно отнести к компьютерам пятого поколения, т.к. они используют новую технологическую базу.
- Фактически, в настоящее время, перехода к новой технологической базе не произошло.
- ЭВМ 5-го поколения должны иметь более развитую периферийную систему, чем современные компьютеры
- ЭВМ 5-го поколения должны различать звуковую, зрительную, сенсорную информацию и обрабатывают ее по законам деятельности человеческого мозга.
- ЭВМ 5-го поколения должны различать звуковую, зрительную, сенсорную информацию и обрабатывать ее по собственным законам, которые они определят в процессе своего развития самостоятельно, без вмешательства человека.