

Министерство науки и высшего образования Российской Федерации
Чайковский филиал
федерального государственного автономного образовательного учреждения
высшего образования
**Пермский национальный исследовательский
политехнический университет**

МП 12.8-2022

МЕТОДИЧЕСКИЕ ПРЕДПИСАНИЯ

к выполнению курсовой работы

по дисциплине

«ЭВМ и периферийные устройства»

Направление: 09.03.01 Информатика и вычислительная техника

Очная форма обучения
Заочная форма обучения

Сухих, И.И. Методические предписания к выполнению курсовой работы по дисциплине «ЭВМ и периферийные устройства» для всех форм обучения направления подготовки 09.03.01 Информатика и вычислительная техника/ И.И. Сухих. – Чайковский, 2022. –78с.

Составитель: ст. преподаватель кафедры автоматизации, информационных и инженерных технологий ЧФ ПНИПУ Сухих И.И.

Методические предписания к выполнению курсовой работы по дисциплине «ЭВМ и периферийные устройства» для всех форм обучения направлений подготовки: 09.03.01 Информатика и вычислительная техника. Методические предписания составлены в соответствии с рабочими программами дисциплины.

©Пермский национальный исследовательский
политехнический университет
Чайковский филиал, 2022
©Сухих И.И., 2022

ВВЕДЕНИЕ

В современном обществе повсеместно внедрены компьютерные технологии. Изучение устройства и принципа работы составляющих элементов электронных вычислительных машин (ЭВМ) является актуальной задачей при подготовке квалифицированных специалистов в высших учебных заведениях.

Дисциплина «ЭВМ и периферийные устройства» посвящена изучению: основных характеристик и области применения ЭВМ различных классов; функциональной и структурной организации центрального процессора; основных стадий выполнения команд центральным процессором; организации прерываний в ЭВМ; организации памяти ЭВМ; организации ввода-вывода; периферийных устройств; архитектурных особенностей параллельных систем; а также многомашинных и многопроцессорных вычислительных систем.

В дисциплине вместе с архитектурой ЭВМ изучается и язык низкого уровня Ассемблер. Это связано с тем обстоятельством, что ЭВМ является исполнителем алгоритма на машинном языке, поэтому знание языка низкого уровня необходимо для понимания архитектуры ЭВМ.

Большую роль в развитии творческих навыков студентов призвано сыграть выполнение курсовой работы, которое закрепит приобретенные ранее знания по дисциплине ЭВМ и периферийные устройства и дает опыт при решении конкретных производственных задач. В процессе выполнения курсовой работы студент приобретает необходимые навыки в работе с научнотехнической и справочной литературой, проектно-конструкторской документацией, нормами, расценками и т.д.

Целью курсовой работы — исследование структуры и характеристик ЭВМ и периферийных устройств, приобретение практических навыков в определении неисправностей периферийных устройств и путей устранения и предотвращения их возникновения, а также ознакомление с функционированием ЭВМ на уровне взаимодействия аппаратных компонентов и получение навыков управления ими, написав и отладив программу на языке Ассемблер.

Задачи курсовой работы:

- описание предмета исследования;
- построение классификации и описание принципа функционирования;
- описание конструктивных особенностей и характеристик различных видов;
- построение классификации неисправностей предмета исследования;
- определение методы диагностики, устранения, а также профилактики возникновения перечисленных неисправностей;
- ознакомление с организацией прерываний в ЭВМ;
- ознакомление с программированием видеоадаптеров в нестандартных режимах;
- построение математической модели трехмерной фигуры;
- проведение алгоритмизации и написание программы установки нестандартных режимов видеоадаптеров.

Выполнение курсовой работы должно способствовать углубленному усвоению лекционного курса и приобретению навыков в области решения производственных задач и разрешения ситуаций. Она базируется на изучении законов, нормативных и методических материалов, литературных источников, а также на практическом материале технологических процессов, экспериментальных и статистических данных.

Структура курсовой работы должна способствовать раскрытию избранной темы и отдельных ее вопросов, и должна быть выполнена в соответствии с данным учебно-методическим пособием.

Учебно-методическое пособие содержит методику выполнения курсовой работы и предназначено для студентов всех форм обучения, обучающихся по направлению подготовки 09.03.01 – «Информатика и вычислительная техника», а также смежных направлений.

1 ОРГАНИЗАЦИЯ ВЫПОЛНЕНИЯ КУРСОВОЙ РАБОТЫ

1.1 Порядок выполнения

Курсовая работа выполняется в соответствии с заданием, выдаваемым каждому студенту. Задание на курсовую работу выдается преподавателем в сроки, установленные учебным расписанием. В задании на курсовую работу указывается тема, перечень подлежащих разработке вопросов и обязательного графического материала, исходные данные к работе и рекомендованная литература. А также указывается дата выдачи задания преподавателем, дата получения задания студентом и дата защиты курсовой работы. Задание на курсовую работу подписывается студентом и преподавателем. Варианты заданий на курсовую работу представлены в приложении А.

Курсовая работа выполняется каждым студентом индивидуально под руководством преподавателя. При выдаче задания на курсовую работу преподаватель разъясняет цели и задачи работы, излагает требования к содержанию и оформлению пояснительной записки и графического материала.

Выполненная и оформленная курсовая работа представляется преподавателю не позднее, чем за неделю до защиты. Преподаватель проверяет соответствие пояснительной записки и графических материалов заданию, правильность и обоснованность принятых технических решений и расчетов, грамотность и логичность изложения материала пояснительной записки, правильность выполнения графического материала. Правильно выполненная и оформленная курсовая работа допускается к защите. Если курсовая работа содержит принципиальные ошибки, недоделки или не соответствует заданию, то она возвращается студенту на доработку. Исправленная курсовая работа представляется преподавателю для повторной проверки.

Защита курсовой работы производится в установленные сроки. Защита курсовой работы состоит из доклада на 10-12 минут с использованием мультимедиа проектора и ответов на вопросы преподавателя. В докладе должна

быть полностью отражена тема курсовой работы.

По результатам защиты выставляется оценка. При выставлении оценки учитывается:

- научно-технический уровень курсовой работы и глубина исследования;
- качество выполнения пояснительной записки и графического материала;
- четкость и логичность изложения доклада, правильность ответов на вопросы;
- степень самостоятельности при работе над курсовой работой.

1.2 Структура курсовой работы

Курсовая работа по дисциплине ЭВМ и периферийные устройства состоит из трех частей:

- Теоретическая часть: теоретические сведения о предмете исследования.
- Практическая часть: практические сведения о неисправностях предмета исследования и методов их устранения.
- Лабораторная часть: работа с прерываниями.

Первая часть курсовой работы заключается в поиске, анализе и обработке информации по указанной теме и представляет собой реферативное сообщение о предмете исследования. Должна содержать сведения, актуальные на сегодняшний день и описывающие предмет исследования с позиций классификации, конструктивных особенностей, принципов функционирования и применения в составе вычислительной техники.

Вторая часть курсовой работы является практической частью и заключается в структурировании сведений о неполадках, которые могут возникнуть в процессе функционирования предмета исследования и методах их

устранения.

Третья часть курсовой работы заключается в ознакомлении с функционированием ЭВМ на уровне взаимодействия аппаратных компонентов, и получить навыки управления ими. Ознакомиться с программированием видеоадаптеров в нестандартных режимах и ознакомиться с организацией прерываний в ЭВМ. Написать и отладить программу на языке программирования с применением встроенного ассемблера.

Пояснительная записка курсовой работы по дисциплине ЭВМ и периферийные устройства включает:

- Титульный лист.
- Задание на курсовую работу.
- Содержание.
- Введение.
- Теоретические сведения о предмете исследования.
- Сведения о неисправностях предмета исследования и их устранении.
- Работа с прерываниями.
- Заключение.
- Список литературы.
- Приложения.

К пояснительной записке обязательно прилагается задание на курсовую работу, которое не включаются в содержание и общее количество страниц пояснительной записки.

Титульный лист

Титульный лист должен содержать наименование темы, фамилию и инициалы студента, наименование группы, фамилию и инициалы преподавателя, год выполнения работы. Пример оформления титульного листа приведен в приложении Б.

Введение

В начале этого раздела нужно обосновывается выбор темы, определяемый

её актуальностью, формулируется её проблема и круг вопросов, необходимых для её решения. Описывается актуальность исследования неполадок возникающих в процессе функционирования предмета исследования методами их устранения. Описывается актуальность работы с периферийными устройствами с использованием прерываний для установки нужных параметров.

В конце раздела формулируются цель курсовой работы и задачи, которые необходимо решить для достижения поставленной цели. Объем «Введения» должен быть 1 - 2 страницы.

Теоретические сведения о предмете исследования

Раздел должен содержать краткие сведения о предмете исследования, включающую в себя историческую справку, современные сведения и тенденции развития.

Необходимо построить разноаспектную классификацию. Описать конструктивные особенности и характеристики различных видов предмета исследования. Выявить сильные и слабые стороны. Сделать сравнительный анализ.

В конце раздела требуется подвести итог проделанному анализу. Описание предмета исследования следует сопровождать иллюстративным материалом: рисунками, схемами, графиками и таблицами. Объем раздела с учетом иллюстративного материала не должен превышать 15 страниц.

Сведения о неисправностях предмета исследования и их устранении

Раздел должен содержать классификацию возможных неисправностей предмета исследования. Описываются предпосылки возникновения неисправностей. Методы диагностики и устранения перечисленных неисправностей. Приводится список профилактических мер, позволяющий отсрочить или избежать возникновения неисправностей.

В конце раздела требуется сделать вывод о надежности предмете исследования. Определить способы повышения надежности. Описание неисправностей следует сопровождать иллюстративным материалом: рисунками,

схемами, графиками и таблицами. Объем раздела с учетом иллюстративного материала не должен превышать 10 страниц.

Работа с прерываниями

Раздел должен содержать формулировку задания. Построение математической модели трехмерной фигуры. Описание организации прерываний в ЭВМ. Описание способов программирования видеоадаптеров в нестандартных режимах. Блок-схема алгоритма программы. Описание программы, а также описание всех ключевых блоков программы. Иллюстрация работы программы. Листинг программ приводится в приложении

В конце раздела требуется сделать вывод по полученным результатам. Объем раздела с учетом иллюстративного материала не должен превышать 10 страниц.

Заключение

В заключение излагаются выводы и предложения, полученные студентом в процессе выполнения курсовой работы. Выводы пишутся в тезисном стиле (по пунктам). В конце заключения необходимо включить фразу «Цели и задачи курсовой работы достигнуты».

Список литературы

Включает список используемых источников в количестве не менее 15. В отчете необходимо привести ссылки на литературу. Список литературы оформляется согласно первому упоминанию источника в отчете по курсовой работе.

Приложения

В приложения включают вспомогательный материал и данные, не вынесенные в основную часть, но являющиеся необходимыми при реализации задачи курсовой работы. Например: схемы, графики, объемные таблицы, исходный код программы, дополнительные иллюстрации.

1.3 Задания на курсовую работу

В теоретической и практической части курсовой работы необходимо

произвести исследование современных устройств компьютерной техники: выявить актуальные сведения и тенденции развития, построить разноаспектную классификацию, выявить конструктивные особенности и характеристики. На основе анализа определить сильные и слабые стороны. Определить классификацию возможных неисправностей предмета исследования и определить предпосылки их возникновения. Описать методы диагностики и устранения перечисленных неисправностей и определить список профилактических мер, позволяющих отсрочить или избежать возникновения неисправностей

Список устройств для исследования:

- Струйные принтеры.
- Лазерные принтеры.
- Светодиодные принтеры.
- Многофункциональные устройства.
- Графопостроители.
- Сканеры.
- ЭЛТ - мониторы.
- Жидко - кристаллические мониторы.
- Графические планшеты.
- Плазменные мониторы.
- Мультимедиа проектор.
- Жесткие магнитные диски.
- Дисковые массивы RAID.
- Твердотельные жесткие диски.
- Акустические системы.
- Звуковые карты.
- Графические адаптеры.
- Аудиосистемы.
- TV-тюнеры и карты видео захвата.

– Системные платы.

Лабораторная часть курсовой работы посвящена разработке программы осуществляющей трехмерное вращение фигуры вдоль оси X под углом 45 градусов к оси Z в плоскости перпендикулярной XOY. Программу необходимо написать на языке Паскаль (Pascal). Установление видеорежима отображения фигуры осуществляется с использованием встроенного языка Ассемблер (Assembler) и библиотеки DOS Turbo Pascal. Список возможных видеорежимов представлен в таблицах 2.14 и 2.15. Вывод точек производится непосредственно в видеопамять. Фигуры для построения

- Пирамида 3 грани.
- Пирамида 4 грани.
- Пирамида 5 граней.
- Пирамида 7 граней.
- Пирамида 9 граней.
- Конус.
- Усеченная пирамида 3 грани.
- Усеченная пирамида 4 грани.
- Усеченная пирамида 5 граней.
- Усеченная пирамида 7 граней.
- Усеченная пирамида 9 граней.
- Усеченный конус.
- Обелиск.
- Цилиндр.
- Усеченный цилиндр.
- Полый цилиндр.
- Усеченный полый цилиндр.
- Сфера.
- Шаровой сегмент.
- Шаровой сектор.

2 МЕТОДИКА ВЫПОЛНЕНИЯ КУРСОВОЙ РАБОТЫ

2.1 Основы языка Ассемблер

Микропроцессор способен выполнять команды, находящиеся в памяти в виде двоичных кодов. Написать программу в двоичной кодировке очень трудно. Языки программирования высокого уровня намного облегчают написание программ, но они не дают доступа ко всем ресурсам компьютера и размер получаемого кода оказывается значительным. Альтернативой, является язык Ассемблер. Ассемблер является регистронезависимым.

Ассемблер (Assembler) - машинно-зависимый компилятор, преобразующий составленные специальным образом текстовые строки в машинные инструкции. Ассемблер упрощает разработку программ за счет того, что позволяет работать с ресурсами компьютера, при этом оперируя понятными разработчику командами, использующими мнемонический код, а в качестве операндов последовательности байтов или слов.

Ассемблер также содержит служебные инструкции, определяющие формат программы и данные. Существуют самостоятельные Ассемблер- системы: транслятор + компоновщик.

Примеры:

Turbo Assembler (TASM) + TLINK;

Macro Assembler (MASM) +LINK;

Flat Assembler (FASM) + компоновщик.

Ассемблер может встраиваться в языки высокого уровня. Такой ассемблер называется встроенным.

Примеры:

Pascal и Object Pascal;

Delphi;

C;

C++.

Встроенный ассемблер позволяет программировать на уровне машинных инструкций. Программируя на встроенном Ассемблере, пишут последовательность машинных инструкций таким образом, чтобы реализовать вычисления с максимальной скоростью при минимальных затратах памяти. Компиляторы языков высокого уровня, например язык Паскаль, неизбежно вносят в машинный код избыточность, которая уменьшает скорость счета и увеличивает затраты памяти. Программирование на уровне машинных инструкций сложное занятие и не может сравниться по скорости разработки программ с программированием на языках высокого уровня. Встроенный ассемблер не предназначен для написания законченных программ.

Регистры центрального процессора

Микропроцессор имеет ячейки памяти образующие сверхбыструю оперативную память, которые называются регистрами. Скорость доступа к регистрам значительно выше, чем к оперативной памяти, но они имеют небольшой объем. Микропроцессором регистры используются для выполнения простейших операций: арифметические и логические операции. Существуют регистры, в которых хранятся адреса оперативной памяти, с которыми оперирует процессор. Существуют базовые регистры, которые в функциональном отношении делятся на пять групп.

Регистры общего назначения (АХ, ВХ, СХ, ДХ)

Предназначены для хранения операндов и выполнения основных команд; любой из них может использоваться как совокупность двух независимых друг от друга 8-разрядных регистров: старшего байта регистра (АН, ВН, СН, ДН) и младшего байта (АL, ВL, СL, DL). Буквы Н и L в их именах происходят от слов HIGH - больший (старший) и LOW - меньший (младший)

15	8 7	0	
АН	AL	АХ - Аккумулятор	
ВН	BL	ВХ - Базовый регистр	
СН	CL	СХ - Счетчик	
ДН	DL	ДХ - Регистр данных	

Регистр AX (*Accumulator eXtended*). Является основным сумматором, используемым во всех арифметических операциях. Только с помощью AX и его полурегистров AH и AL возможен обмен данными с портами ввода/вывода.

Регистр BX (*Base eXtended*). Сумматор в арифметических операциях, а также как базовый регистр при индексной адресации.

Регистр CX (*Counter eXtended*). Счетчик при выполнении операций повторения и сдвига, также участвовать в арифметических операциях.

Регистр DX (*Data eXtended*). Регистр данных в операциях ввода/вывода, а также сумматор при обработке длинных целых чисел (32-разрядных).

Регистры указатели (SP, BP, IP)

Регистры указатели служат для работы со стеком для указания смещения при адресации памяти. Стек - это участок автоматически адресуемой памяти, используемый для временного хранения данных, для передачи параметров вызываемым подпрограммам и для сохранения адреса возврата при вызове процедур и прерываний. Регистры являются 16 разрядными (рисунок 2.18).

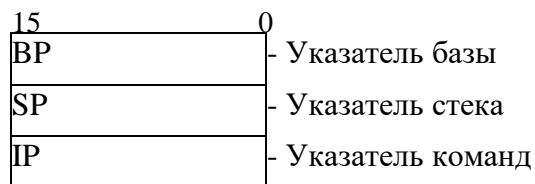


Рисунок 2.18 - Регистры указатели

Регистр BP (*Base Pointer*). Указатель базы. Облегчает создание и использование локального стека для использования внутри процедуры.

Регистр SP (*Stack Pointer*). Указывает на вершину стека. Вместе с регистром SS адресует ячейку памяти, куда будет помещаться операнд или откуда он будет извлекаться. Содержимое регистра автоматически уменьшается после размещения в стеке очередного операнда и увеличивается после извлечения операнда из стека.

Регистр IP (*Instruction Pointer*). Указывает на адрес команды в сегменте

кода. Определяет смещение относительно начала сегмента кода CS очередной исполняемой машинной инструкции. Напрямую содержимое регистра IP изменить нельзя. Регистр IP автоматически изменяется в ходе исполнения инструкции, обеспечивая правильный порядок выборки команд из памяти. При выполнении обычных команд значение IP увеличивается на размер выполненной команды. Содержимое регистра IP можно изменить специальными командами передачи управления.

Индексные регистры (SI, DI)

Применяются для индексной адресации (рисунок 2.19), для использования в операциях сложения, вычитания и для операций над байтовыми строками. Байтовая строка в языке Ассемблер представляет собой ряд последовательных байт.

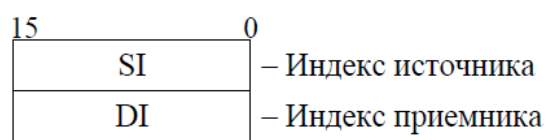


Рисунок 2.19 – Индексные регистры

Регистр SI (*Source Index*). Определяет адрес источника информации при индексной адресации данных, как правило используется вместе с регистром DS.

Регистр DI (*Destination Index*). В паре с регистром SI определяет приемник информации при межсегментном обмене данными.

Сегментные регистры (CS, DS, SS, ES)

Используются для сегментной адресации. Код, данные и стек находится каждый в своем сегменте. Регистры являются 16 разрядными (рисунок 2.20).

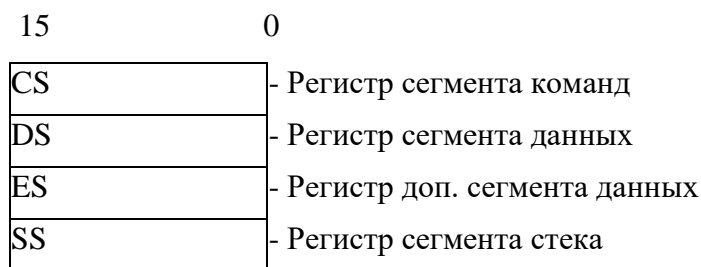


Рисунок 2.20 - Сегментные регистры

Регистр CS (*Code Segment*). Хранит номер сегмента памяти и кода, в котором содержится текущая машинная инструкция. Для получения полного адреса следующей команды его содержимое сдвигается влево на 4 разряда и складывается с регистром-указателем IP. Содержимое CS автоматически изменяется в командах межсегментного перехода и вызова процедур.

Регистр DS (*Data Segment*). Хранит номер сегмента памяти и данных, в котором содержится данные (константы и переменные). Все глобальные переменные и типизированные константы языка Pascal всегда располагаются в единственном сегменте, адресуемом этим регистром.

Регистр ES (*E Segment*). Используется для межсегментного обмена данными и в некоторых строковых операциях.

Регистр SS (*Stack Segment*). Хранит номер сегмента стека. Определяет стартовый адрес сегмента, в который помещается стек для программы. По умолчанию смещения для сегмента стека задаются в регистрах SP и BP.

Регистр флагов (CF, PF, AF, ZF, SF, TF, IF, DF, OF)

Используется для хранения признаков состояния процессора. Состоит из 16 бит, из них используются только 9 ([рисунок 2.21](#)).

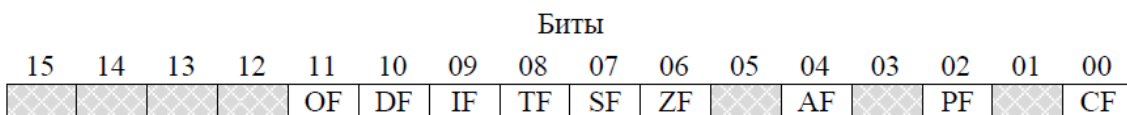


Рисунок 2.21 – Регистры флагов

Биты регистра состояния устанавливаются или очищаются в зависимости от результата исполнения предыдущей команды и используются некоторыми командами процессора. Биты регистра состояния могут также устанавливаться и очищаться специальными командами процессора.

Флаг переноса CF (*Carry Flag*). Содержит 1, если произошел перенос единицы при сложении или заем единицы при вычитании. Используется в циклических операциях и операциях сравнения.

Флаг четности PF (*Parity Flag*). Содержит 1, если в результате операции

получено число с четным количеством значащих разрядов. Используется в операциях обмена для контроля данных.

Флаг внешнего переноса *AF (Auxiliary Flag)*. Контролирует перенос из 3-го бита данных, полезен при операциях над упакованными десятичными числами.

Флаг нуля *ZF (Zero Flag)*. Содержит 1, если в результате операции получен ноль, и равен 0 в противном случае.

Флаг знака *SF (Sign Flag)*. Содержит 1, если в результате операции получено отрицательное число (с единицей в старшем разряде).

Флаг трассировки *TF (Trace Flag)*. Содержит 1, если программа выполняется по шагам, с передачей управления после каждой выполненной команды по прерыванию с вектором 1.

Флаг прерываний *IF (Interrupt Flag)*. Содержит 1, если микропроцессору разрешена обработка прерываний. Установка этого флага в 1 приводит к тому, что процессор перестает обрабатывать прерывания от внешних устройств. Устанавливают на короткое время для выполнения критических участков кода.

Флаг направления *DF (Direction Flag)*. Управляет направлением передачи данных: если он содержит 0, то после каждой индексной операции содержимое индексных регистров увеличивается на 1, в противном случае - уменьшается на 1.

Флаг переполнения *OF (Overflow Flag)*. Содержит 1, если в результате операции получено число, выходящее за разрядную сетку микропроцессора.

Содержимое регистров и флагов можно посмотреть в Turbo Pascal выбрав в меню Debug (*Отладка*) пункта Register (*Регистры*). Чтобы отследить изменение содержимого регистров и флагов необходимо поэтапно выполнять программу ([рисунок 2.22](#)). Необходимо выбрать в меню Run (*Пуск*) пункт Step over (*Обойти программу*) или нажать на клавиатуре кнопку F8.

CPU 3			
AX	000A	DX	10C3
CX	0468	BX	000F
IP	0017	CS	10BC
SI	026F	DS	114E
DI	015A	ES	114E
SP	3FFE	SS	1179
BP	3FFE		
c=1	z=0	s=1	o=0
p=0	i=1	a=1	d=0

Рисунок 2.22 -
Окно регистры

Описанное ранее описание архитектуры x86 микропроцессора является

базовым для языка Ассемблер, в том числе и для встроенного ассемблера. Язык Ассемблер содержат дополнительные возможности, которые облегчают разработку программ. Дополнительные возможности отражаются в директивах и макрокомандах языка Ассемблера.

Во встроенном ассемблере почти полностью отсутствуют средства описания переменных и данных, т.к. эти объекты описываются средствами языка в который встраивается ассемблер. Сегментные регистры должны иметь точно такие же значения к моменту завершения работы ассемблерного оператора. Регистры после завершения работы ассемблерного оператора могут иметь произвольное значение. Исключением является ассемблерные функции, использующие некоторые регистры для возврата своего значения

Команды языка Ассемблер

Ассемблерными командами будем называть команды на языке встроенного ассемблера, вставляемые в тело программы операторами `asm . . . end`.

Оператор `asm . . . end`

Оператор `asm . . . end` является операторными скобками. Слово `asm` активизирует встроенный ассемблер и ограничивается ближайшим по тексту словом `end`. Оператор `asm . . . end` располагается только внутри тела программы или подпрограммы. Тело ассемблерного оператора `asm . . . end` может быть как пустым так и содержать несколько ассемблерных команд.

```
asm
```

```
<Команды встроенного ассемблера> end;
```

Каждая ассемблерная команда должна пишется в отдельной строке или могут отделяться друг от друга символом «;». В конце строки, содержащей ассемблерную команду можно использовать комментарии, который оформляются по правилам языка Pascal, и должны ограничиваться символами «{», «}» или «(*», «*)». Комментарии внутри ассемблерных команд не допускаются.

Структура ассемблерной команды

Типовая форма структуры ассемблерной команды имеет вид: [Метка]
[Префикс] [Код [Операнд [,Операнд]]]

В квадратных скобках указываются необязательные элементы структуры.

Метки

Перед ассемблерной командой может находиться метка. В ассемблере используется два типа меток: глобальные и локальные.

Глобальная метка - метка языка Pascal и объявляется после зарезервированного слова Label в разделе описаний. Используя такую метку можно передать управление в тело ассемблерного оператора оператором GOTO.

Локальные метки объявляются в теле ассемблерного оператора. Такие метки обязаны начинаться символом «@» и заканчиваться «:». Символ «@» нельзя использовать в именах языка Pascal. Локальная метка известна только внутри ассемблерного оператора. В разных ассемблерных операторах можно использовать одноименные локальные метки.

Пример:

```
asm
    @B: {Локальная ассемблерная метка}
end;
```

Префиксы

Встроенный ассемблер поддерживает следующие префиксы команд:

Таблица 2.1 - Префиксы команд

Мнемонический код	Пояснение
LOCK	Захват шины
REP/REPE/REPNE/REPZ/REPZ	Повтор строковой команды
SEGCS/SEGDS/SEGSS/SEGES	Определяют сегментные регистры

Префиксы SEGCS/SEGDS/SEGSS/SEGES определяют сегментные регистры, которые необходимо использовать вместо сегментных регистров и распространяются только на следующие за ними ассемблерные команды. Если префикс указан без кода инструкции, он распространяет свое действие на следующую ассемблерную команду.

Операнды

Операндами встроенного ассемблера могут быть выражения, которые состоят из: регистров, констант, имен и символов операций. Ассемблер

поддерживает как строковые, так и числовые константы. Строковые константы заключаются в апострофы «'» или кавычки «"».

Примеры:

'Строковая константа, заключенная в апострофы'

"Строковая константа, заключенная в кавычки"

В ассемблерной константе, объявленной с помощью кавычек, символ апостроф рассматривается наравне с другими символами. Если внутри константы необходимо указать апостроф, он удваивается.

Числовые константы могут быть только целыми, размер константы не может быть больше двойного слова (4 байт). Запись в числовую константу используется десятичная форма, но ассемблер поддерживает двоичные, восьмеричные и шестнадцатеричные константы.

Двоичная константа составляется как комбинация единиц и нулей, заканчивающаяся символом В (Binary - двоичный); при записи восьмеричной константы используются символы 0..7, а в ее конце ставится символ О (Octal - восьмеричный); шестнадцатеричная константа записывается по правилам языка Паскаль (начинается с символа #) либо по правилам Турбо Ассемблера (TASM): начинается с цифры, в конце ставится символ Н (Hexadecimal - шестнадцатеричный).

Локальные метки являются единственными именами, которые разрешается определять внутри ассемблерного оператора. Константы, переменные и подпрограммы - должны определяться только с помощью средств языка Pascal.

Область определения имен подчиняется тем же правилам, что и в языке Pascal - имена должны быть видимы в том месте, где они используются, и они локализируются в пределах блока, в котором описаны.

Во встроенном ассемблере используются три предопределенных имени:

@Code - текущий сегмент кода @Data - начальный сегмент данных @Result - ссылка внутри функции на ее результат

Ассемблерные выражения имеют тип, но определяет только размер объекта в памяти и не ограничивает применяемые к нему операции. Встроенный ассемблер имеет следующие predetermined типы приведенные в [таблице 2.2](#). Имена predetermined типов можно использовать для приведения типов выражений.

Таблица 2.2 - Предопределенные типы

Тип	Длина в памяти	Тип	Длина в памяти
BYTE	1	TBYTE	10
WORD	2	NEAR	-
DWORD	4	FAR	-
QWORD	8		

Имена не могут использоваться в операндах встроенного ассемблера:

- стандартные процедуры и функции (например, WriteLn, Chr);
- predetermined массивы Mem, MemW, MemL, Port, PortW;
- константы с плавающей точкой, строковые и множественного типа;
- символ @Result вне функции

Основные команды языка Ассемблер

Форматы условного написания команд зависят от типа используемого процессорами операндов, входящих в команду:

1. Двухадресная команда состоит из кода операции и двух операндов. В качестве операндов могут выступать выражения;
2. Одноадресная команда (код операции и операнд);
3. Безадресная команда (код операции, неявно меняет содержимое регистров).

Для удобства пользования все команды разделены на 6 функциональных групп: пересылки данных, арифметические, битовые, передачи управления, строковые, прерываний, управления. Внутри каждой группы команды объединяются в подгруппы по общим дополнительным признакам

Команды пересылки данных

Команды пересылки данных обычно наиболее часто используются из

всего набора команд любой ЭВМ. Большая часть каждой задачи по обработке данных заключается в переносе информации из одного места в другое.

Таблица 2.3 - Команды пересылки данных

Команда	Формат	Пояснение
<i>Команды общего назначения</i>		
MOV	MOV <u>приемник, источник</u>	Переслать значение
PUSH	PUSH <u>источник</u>	Поместить в стек
POP	POP <u>приемник</u>	Извлечь из стека
XCHG	XCHG <u>приемник, источник</u>	Обменять значения
XLAT	XLAT <u>таблица</u>	Загрузить в AL байт из таблицы
<i>Команды ввода-вывода</i>		
IN	IN <u>аккумулятор, порт</u>	Читать из порта
OUT	OUT <u>порт, аккумулятор</u>	Записать в порт
<i>Команды пересылки адреса</i>		
LEA	LEA <u>регистр 16, память 16</u>	Загрузить исполнительный адрес
LDS	LDS <u>регистр 16, память 32</u>	Загрузить в DS: регистр 16 адрес
LES	LES <u>регистр 16, память 32</u>	Загрузить в ES: регистр 16 адрес
<i>Команды пересылки флагов</i>		
LAHF	LAHF	Загрузить флаги в AH
SAHF	SAHF	Установить флаги из AH
PUSHF	PUSHF	Поместить флаги в стек
POPF	POPF	Извлечь флаги из стека

Команда MOV позволяет в защищенном режиме переслать байт или слово из регистра в регистр, из памяти в регистр или из регистра в память, но нельзя пересылать данные из памяти в память. Тип пересылаемых данных (байт или слово) определяется регистром, участвующим в пересылке.

Примеры:

- mov AX, ABC {Пересылка данных из памяти в AX}
- mov ABC, AH {Пересылка данных из AH в память}
- mov DS, BX {Пересылка из BX в сегмент данных}
- mov CH, 20 {Пересылка константы в регистр}
- mov ABC, \$FF {Пересылка константы в память}
- mov Mem0, Mem1 {Нельзя занести данные из памяти в память}

Примеры:

- mov AX, Mem1 {В AX можно занести данные из памяти}
- mov Mem0, AX {В память Mem0 занести данные из AX}
- mov DS, X {В DS нельзя занести константу/переменную}
- mov AX, X {Нужно занести сначала в регистры POH}

Примеры:

- mov DS, AX {Нельзя один сегментный регистр}
- mov ES, DS {занести в другой сегментный регистр}
- mov AX, DS {В AX занести данные из регистра DS}
- mov EX, AX {Занести в регистр ES данные из AX}

Команды PUSH и POP широко используются для временного сохранения регистров и данных, а также для обмена значениями между регистрами. Каждая из них работает только со словом, т.е. в стек нельзя поместить или извлечь из него одиночный байт.

Команда PUSH служит для занесения содержимого 16-битного машинного слова в стек. Источником могут быть регистры общего назначения, индексный регистр, сегментный регистр, ячейка памяти. Сначала уменьшается на 2 содержимое указателя SP, а затем операнд помещается по адресу SS : SP.

Примеры:

- push 7 {Поместить 7 в стек}
- push AX {Поместить AX в стек}
- push DS {Поместить DS в стек}

Команда POP извлекает 16-битный операнд из стека и пересылает его в место назначения, указанное в команде - регистры общего назначения, индексный регистр, сегментный регистр, ячейку памяти. После извлечения

операнда из стека автоматически формирует новый адрес вершины стека, выполняя действие. При извлечении из стека сначала читается память по адресу $SS:SP$, а затем SP увеличивается на 2.

Примеры:

```
pop DX {Поместить значение из стека в DX}
```

```
pop ES {Поместить значение из стека в ES}
```

Команда загрузки адреса `LEA` загружает в регистр адрес смещение нужного участка памяти. Этого же можно сделать с помощью зарезервированного слова `OFFSET`, стоящего перед именем переменной.

Пример:

```
var Y: Word;
```

```
asm
```

```
mov AX, OFFSET Y {Загружает смещение Y в AX}
```

```
lea AX, Y {Загружает смещение Y в AX}
```

```
end;
```

Команды `LEA` разрешает использовать индексную адресацию и используется при пересылке массивов данных. Команды адресной загрузки `LDS` и `LES` загружают первое 16-битное слово из источника в регистр-приемник, а затем следующее слово - в регистр `DS` или `ES`, они рассчитаны на загрузку полного адреса операнда (сегмент и смещение).

Команды `LAHF`, `SAHF`, `PUSHF` и `POPF` служат для работы с флагами. Команды `LAHF` и `SAHF` служат для установки и загрузки значений регистров флагов. Содержимое регистров флагов можно сохранить в стек с помощью команды `PUSHF` и восстановить из стека с помощью команды `POPF`.

Арифметические команды

Микропроцессоры архитектуры x86 используют четыре арифметические действия: сложение, вычитание, умножение, деление над 8- и 16-битными данными со знаком и без знака, представляющими целые числа. Работа с дробными числами осуществляется с применением математического

сопроцессора x87. Арифметические команды влияют на флаги в регистре FLAGS, команды сложения и вычитания устанавливают флаги SF, ZF, PF, AF, CF и OF по результатам своих действий; команды умножения устанавливают, только флаги CF и OF, которые устанавливаются в 1, если старшая половина результата отлична от нуля; при выполнении команд деления флаги принимают произвольные значения, и их обычно не используют.

Таблица 2.4 - Арифметические команды

Команда	Формат	Комментарий
<i>Команды сложения</i>		
ADD	ADD <u>приемник, источник</u>	Сложить
ADC	ADC <u>приемник, источник</u>	Сложить, добавить перенос
INC	INC <u>приемник</u>	Увеличить на единицу
<i>Команды вычитания</i>		
SUB	SUB <u>приемник, источник</u>	Вычесть
SBB	SBB <u>приемник, источник</u>	Вычесть с заемом
DEC	DEC <u>приемник</u>	Уменьшить на единицу
NEG	NEG <u>приемник</u>	Обратить знак
CMP	CMP <u>приемник, источник</u>	Сравнить
<i>Команды умножения</i>		
MUL	MUL <u>источник</u>	Умножить без знака.
IMUL	IMUL <u>источник</u>	Умножить со знаком.
<i>Команды деления</i>		
DIV	DIV <u>источник</u>	Делить без знака.
IDIV	IDIV <u>источник</u>	Делить со знаком.
<i>Команды расширения знака</i>		
CBW	CBW	Преобразовать байт в слово
CWD	CWD	Преобразовать слово в двойное слово

Беззнаковые числа для представления значения используются все биты и они могут быть сохранены в переменных типа Byte и Word. Знаковые числа в старшем разряде хранят знак числа и они могут быть сохранены в переменных типа ShortInt и Integer. Двоично-десятичные числа используют по 4 бита для каждого десятичного разряда и могут быть запакованными или не запакованными. В запакованных данных один байт хранит 2 десятичные цифры,

где старшая цифра храниться в старшем полубайте. В не упакованных данных один байт хранит 1 десятичную цифру, где старший полубайт не используется. Основные арифметические команды микропроцессора (ADD, SUB, MUL, DIV) не учитывают двоично-десятичную форму представления чисел, из-за этого в архитектуру микропроцессора включены команды коррекции результата.

Команда ADD служит для сложения двух чисел, как со знаком, так и без знака. Операнды должны иметь одинаковый размер. Нельзя складывать 16- и 8-битные значения. Результат помещается на место первого операнда. После выполнения команды изменяются флаги, по которым можно определить характеристики результата:

- флаг CF устанавливается, если при сложении произошёл перенос из старшего разряда. Для беззнаковых чисел означает, что произошло переполнение и результат получился некорректным.
- флаг OF обозначает переполнение для чисел со знаком.
- флаг SF равен знаковому биту результата для чисел со знаком, а для беззнаковых он равен старшему биту.
- флаг ZF устанавливается, если результат равен 0.
- флаг PF признак чётности, если результат содержит нечётное число единиц.

Примеры:

```
add BX, 7 {BX = BX + 7}
```

```
add AX, CX {AX = AX + CX}
```

```
add DX, BL {Ошибка, не равный размер операндов}
```

Команда SUB служат для вычитания. Результат помещается на место первого операнда, и выставляются флаги, как и при операции сложения (ADD). Вычитание в микропроцессоре реализовано с помощью сложения.

Примеры:

```
sub AX, BX {AX = AX - CX}
```

```
sub BX, 60 {BX = BX - 60}
```

Команда DEC служит для уменьшения на единицу операнда.

Команда NEG служит для изменения знака второго операнда на противоположный.

Команда CMP служит для сравнения операндов. Команда аналогична команде SUB, но в отличие от нее приемник не изменяется, а получаю флаги, которые затем анализируются командами условного перехода.

Команда MUL служит для умножения чисел без знака. У команды только один операнд — второй множитель, который должен находиться в регистре или в памяти. Хранение первого множителя и результата задается неявно и зависит от размера операнда ([таблица 2.5](#)).

Таблица 2.5 - Операнды команды MUL

Размер операнда	Множитель	Результат
Байт	AL	AX
Слово	AX	DX:AX

В десятичной системе умножая двухзначное число на двухзначное, в результате будет максимум четырехзначное число. Запись «DX:AX» означает, что старшее слово результата будет находиться в DX, а младшее - в AX.

Примеры:

```
mul CL {AX = AL * CL} mul AX {DX:AX = AX * AX}
```

Если старшая часть результата равна нулю, то флаги CF и OF будут иметь нулевое значение и старшая часть результата можно отбросить.

Команда IMUL предназначена для умножения чисел со знаком. В качестве операнда указывается множитель. Местоположение другого множителя и результата определяется по [таблице 2.4](#). Флаги CF = OF = 0, если произведение помещается в младшей половине результата, иначе CF = OF = 1.

Пример:

```
imul BL {с одним операндом AX = AL * BL}
```

Команда DIV предназначена для деления целых двоичных чисел с остатком. Размер делителя, частного и остатка должен быть в 2 раза меньше размера делимого. У команды один операнд — делитель, который должен находиться в регистре или в памяти. Местоположение делимого, частного и остатка задаётся неявно и зависит от размера операнда ([Таблица 2.6](#)).

Таблица 2.6 - Операнды команды DIV

Размер операнда (делителя)	Делимое	Частное	Остаток
Байт	AX	AL	AH
Слово	DX:AX	AX	DX

При работе команды DIV может возникнуть ситуации: когда делитель равен нулю; когда частное не помещается в отведенную ячейку памяти; при делении слова на байт частное больше 255.

При использовании операций деления (DIV, IDIV) в качестве приемника используется регистры AX:DX. Перед тем как произвести деление необходимо обнулить содержимое регистра DX, это можно сделать несколькими способами: `xor DX, DX` или `mov DX, 0`.

Примеры:

`div CL {AL = AX / CL, остаток в AH}`

`div BL {AX = DX:AX / BL, остаток в DX}`

Команда IDIV служит для деления целых двоичных чисел со знаком и с остатком. Операндом является делитель. Местоположение делимого и частного определяется как и для команды DIV. При использовании команды IDIV тоже может возникнуть ситуация с делением на ноль и слишком большим частным.

Примеры:

`idiv CL {AL = AX / CL, остаток в AH}`

`idiv BX {AX = DX:AX / BX, остаток в DX}`

*Пример: Вычислить выражение $25/5 + 2*12 - 65$ результат занести в регистр DX и вывести на экран.*

```

Program sample;
Var rez: Integer;
Begin
asm
    mov AL, 25 {AL = 25}
    mov BL, 7 {BL = 7}
    div BL {AL = AX / BL, остаток в AH}
    mov BX, AX {Занесение в BX результата 25/5}
    mov AL, 2 {AL = 2}
    mov DL, 12 {DL = 12}
    mul DL {AX = AL * DL}
    add BX, AX {AX = BX + AX; 25/5 + 2*12}
    sub AX, 65 {AX = AX - 65}
    mov DX, AX
    mov rez, DX {Занесение результата в переменную}
end;
writeln(rez);
End.

```

Асс

Битовые команды

Битовые команды применяются при исчислении логических выражений и когда необходимо изменить отдельные разряды операнда. Команды выполняются поразрядно над своими операндами.

Таблица 2.7 - Битовые команды

Команда	Формат	Комментарий
<i>Логические команды</i>		
AND	AND <u>приемник, источник</u>	Выполнить AND
OR	OR <u>приемник, источник</u>	Выполнить OR
XOR	XOR <u>приемник, источник</u>	Выполнить XOR
NOT	NOT <u>приемник</u>	Выполнить NOT
TEST	TEST <u>приемник, источник</u>	Проверить
<i>Сдвиговые команды</i>		
SAL/SAR	SAL /R <u>приемник, счетчик</u>	Арифметический сдвиг влево/вправо
SHL/SHR	SHL/R <u>приемник, счетчик</u>	Логический сдвиг влево/вправо
ROL/ROR	ROL/R <u>приемник, счетчик</u>	Циклически сдвиг влево/вправо
RCL/RCR	RCL/R <u>приемник, счетчик</u>	Циклически сдвиг влево/вправо с переносом флага

Команда AND служит для выполнения операции логического И и для обнуления определённых битов числа. Второй операнд называется маской. Обнуляются биты операнда, которые в маске равны 0, значения остальных бит

сохраняются. Команда AND используется для быстрого вычисления остатка от деления на степень 2.

Примеры:

```
and AX, CX {AX = AX & CX}
and BL, 01111111b {Обнуление старшего бита BL}
and DL, 11110000b {Обнуление младшей тетрады DL}
and AX, 11b {AX = остаток от деления AX на 4}
```

Команда OR служит для выполнения операции логического ИЛИ и для установки 1 в определённых бит числа. Если бит маски равен 1, то бит результата будет равен 1, остальные биты сохранят свои значения.

Примеры:

```
or AL, CL {AL = AL | CL}
or CL, 00100101b {Включить биты 0, 2, 5 CL}
```

Команда XOR служит для выполнения операции исключающего ИЛИ и для инверсии бит которые в маске равны 1, остальные сохраняют свое значение. Команда XOR также используется для обнуления регистров. В этом случае необходимый операнд указывается в команде дважды: в качестве источника и в качестве получателя. Если операнды равны, то результат операции всегда равен 0. Обнуление с помощью команды XOR работает быстрее и, в отличие от команды MOV, не содержит непосредственного операнда, поэтому команда получается короче.

Примеры:

```
xor DI, SI {DI = DI ^ SI}
xor BL, 00001111b {Инверсия младшей тетрады BL}
xor BX, BX {Обнуление регистра. mov BX, 0}
```

Команда NOT предназначена для выполнения операции логическое НЕ (инверсия). Каждый бит операнда меняет свое значение на противоположное. Результат помещается на место операнда.

Примеры:

```
mov AL, DAh {AL = 110110102 }
not AL {AL = 001001012 = 2516}
```

Команды AND, OR, XOR и NOT эквивалентны операциям языка Pascal в случае, когда операндами являются целочисленные выражения.

Команда TEST служит для поразрядного суммирования AND, при этом не изменяя значения операндов, устанавливая флаги в соответствии со значением

результата сравнения: обнуляет CF и OF, изменяет PF, ZF, SF и не меняет AF. Если оба операнда содержат по единице, хотя бы в одном соответствующем разряде, флаг ZF установится в 1.

Команды SAL/SAR и SHL/SHR предназначены для сдвига арифметического операнда влево/вправо и сдвига логического операнда влево/вправо соответственно. Используется для работы со знаковыми целочисленными данными. Может использоваться для ускорения арифметических операций деления и умножения. Операция SAR корректно работает с положительными числами и не всегда с отрицательными нечетными числами. Вытесненные в ходе выполнения этих команд, значащие разряды теряются. Необходимо всегда проверять результат, потому что возможен некорректный результат особенно для чисел меньших нуля. Поэтому команды арифметического сдвига применяются достаточно редко.

Примеры:

```
mov BL, 1 {Загрузка в BL единицу}
shr BL, 1 {Сдвиг вправо, на 1 разряд}
```

Пример: Вычислить $j = 16 \cdot x$ и $z = k / 2$ для знаковых 16-битных данных.

Команды сдвига эквивалентны операциям на языке Pascal. $y := x$

```
shl 4; {y:=16*x;}
z:= k shr 1; {z:= k div 2;}
```

На языке Ассемблер.

```
mov AX, x {Загрузка в AX переменную X}
mov BX, k {Загрузка в AX переменную k}
mov CL, 4 {Счетчик сдвига – количество сдвигов}
sal AX, CL {Сдвиг влево, 16*x;}
mov y, AX {Занесение результата сдвига в переменную}
sar BX, 1 {Сдвиг вправо, на 1 разряд (k/2)}
mov z, BX {Занесение результата сдвига в переменную}
```

Команды ROL, ROR, RCL, RCR похожи на SAL, SAR, SHL, SHR, но отличаются тем, что сохраняют значения сдвигаемых разрядов. Команды циклического сдвига широко используются при проверке участка памяти или всей памяти целиком.

Команды ROL и ROR предназначены для простого циклического сдвига влево/вправо, а RCL и RCR предназначены для циклического сдвига через флаг переноса. Инструкции ROL/ROR используются для переупорядочивания бит в байте или в слове.

Пример:

```
mov SI, 49F1h {Загрузка в SI значение}
mov CL, 4 {Счетчик сдвига – количество сдвигов}
ror SI, CL {В SI будет записано значение 149Fh:
           биты 3-0 переместятся в биты 15-12,
           биты 7-4 – в биты 3-0 и т.д.}
```

Инструкции RCR/RCL используются для сдвига операнда, состоящего из нескольких слов.

Пример: Умножить значения в DX:AX размером в двойное слово, на 4

```
shl AX, 1 {15 бит AX сдвигается во флаг переноса}
rcl DX, 1 {Флаг переноса сдвигается в бит 0 DX}
shl AX, 1 {15 бит AX сдвигается во флаг переноса}
rcl DX, 1 {Флаг переноса сдвигается в бит 0 DX}
```

Команды передачи управления

Рассматриваемые ранее команды являются линейными, т.е. команды выполнялись последовательно друг за другом. Регистр IP знает, какая команда выполняется следующей. Команды передачи управления позволяют нарушить естественную последовательность выполнения команд путем изменения содержимого регистра IP.

Рассмотрим базовые команды передачи управления, к ним относятся - команды безусловного перехода на определенный адрес в памяти компьютера, команды условного перехода в зависимости от результата проверки условия и команды организации циклических вычислений.

Команды безусловного перехода

Безусловный переход — это переход, который выполняется всегда. Команды безусловных переходов CALL, RET, JMP.

Таблица 2.8 - Безусловные команды передачи управления

Команда	Формат	Комментарий	Флаг
CALL	CALL <u>имя процедуры</u>	Войти в процедуру	Изменяются только при переключении задачи
RET	RET <u>число</u>	Вернуться из процедуры	
JMP	JMP <u>метка</u>	Перейти	

Команда JMP служит для организации безусловного перехода. Единственный

операнд команды JMP может быть адресом или меткой, регистром или ячейкой памяти, содержащей адрес.

Примеры:

```
jmp @met {Переход на метку @met } jmp DX {Переход по адресу в DX}
```

Команда CALL служит для вызова ассемблерных процедур. Сначала в стек помещается адрес следующей за CALL инструкции либо адрес возврата в стек, потом адрес точки входа в процедуру помещается в регистр IP и сразу за командой CALL будет исполняться уже первая команда процедуры. Адреса точки входа и возврата будут 16-битовыми смещениями для внутрисегментного вызова или 32-битовыми полными адресами для межсегментного.

Команда RET по действию является обратной команде CALL и обеспечивает возврат управления вызывающей программе, адрес возврата берется с вершины стека. При выходе из дальней процедуры извлекаются из стека оба 16-разрядных слова и помещает первое в IP, а второе в CS, а при выходе из ближней извлекаются из стека только смещение и помещает его в IP.

Команды условных переходов

Условные переходы выполняются при определенном условии заданном флагами процессора. Содержимое флагов изменяются после выполнения арифметических и логических команд. Если условие не выполняется, то управление переходит к следующей команде. Команды условных переходов способны передавать управление на ближайшую метку.

Условные переходы служат для реализации ветвлений в программах на основе проверки флагов CF, PF, SF, ZF, OF перед ними располагаются команды CMP и TEST, изменяющие эти флаги.

Базовые команды условного перехода состоят из нескольких букв: первая буква команды J (от JMP - прыжок), остальные буквы в сокращенном виде описывают условие переходы:

- E (*Equal*) - равно;
- N (*Not*) - не, отрицание;
- G (*Greater*) - больше, применяется для чисел со знаком;
- L (*Less*) - меньше, применяется для чисел со знаком;
- A (*Above*) - выше/больше, применяется для чисел без знака;
- B (*Below*) - ниже/меньше, применяются для чисел без знака.

Таблица 2.9 - Условные команды передачи управления

Команда	Формат	Комментарий	Флаг
<i>Любые числа и коды</i>			
JE	JE метка	Перейти, если приемник = источник	ZF=1
JNE	JNE метка	Перейти, если приемник \neq источник	ZF=0
<i>Знаковые числа</i>			
JL/JNGE	JL метка	Перейти, если приемник < источник	SF<>OF
JLE/JNG	JLE метка	Перейти, если приемник <= источник	SF<>OF или ZF=1
JG/JNLE	JG метка	Перейти, если приемник > источник	SF=OF или ZF=0
JGE/JNL	JGE метка	Перейти, если приемник >= источник	SF=OF
JS	JS метка	Перейти, если знак равен 1	SF=1
JNS	JNS метка	Перейти, если знак равен 0	SF=0
JNP	JNP метка	Перейти, если 1 бит нечетный	PF=0
JP	JP метка	Перейти, если 1 бит четный	PF=1
<i>Беззнаковые числа</i>			
JB/JNAE	JB метка	Перейти, если приемник < источник	CF=1
JBE/JNA	JBE метка	Перейти, если приемник <= источник	CF=1 или ZF=1
JA/JNBE	JA метка	Перейти, если приемник > источник	CF=0 и ZF=0
JAЕ/JNB	JAЕ метка	Перейти, если приемник >= источник	CF=0
JNO	JNO метка	Перейти, если нет переполнения	OF=0
JO	JO метка	Перейти, если перенос	OF=1
JCXZ	JCXZ метка	Перейти, если CX=0. Для циклов.	CX=0

Разница в командах перехода для знаковых и беззнаковых данных заключается в том что они реагируют на разные флаги, для знаковых данных используется флаг SF, а для беззнаковых CF.

Примеры:

```
cmp AX, 10 {Проверка AX >= 10?}
jae @metka1 {Если Да, переходим на метку @metka1}
cmp BX, -15 {Проверка BX < -15?}
j1 @metka2 {Если Да, переходим на метку @metka2}
```

Пример: Вычислить заданное условное целочисленное выражение:

Пример: Вычислить заданное условное целочисленное выражение:

$$y = \begin{cases} b/a+10, & \text{если } a \leq b \\ a-5, & \text{если } a = b \\ a^2-3, & \text{если } a > b \end{cases} \quad (2.54)$$

```
Program sample;
Var
    a,b: Word; {Знаковые переменные}
    f: Byte; {Флаг деление на ноль}
    rez: Integer; {Переменная хранения результата}
Begin
    a:=5; b:=7; {Присвоение переменным a, b значений}
asm
    mov AX, a {Занесение в AX переменной a}
    mov BX, b {Занесение в AX переменной b}
    cmp AX, BX {Сравнение a и b}
    jle @metka1 {Условие a<=b}
    jg @metka2 {Условие a>b} {Если a=b} sub AX, 5 jmp
@exit @metka1: {Если a<=b}
    {Проверка деления на ноль}
    cmp AX, 0 {Сравнение AX с 0}
    je @error {Если b=0 переход на метку @error}
    mov AX, BX {AX=BX}
    cwd {Преобразование слово в двойное слово}
    xor DX, DX {Обнуление содержимого регистра DX}
    idiv a {AX= DX:AX/a}
```

```

    add AX, 10 {b/a+10}
    jmp @exit
@metka2: {Если a>b}
    imul a {DX:AX=a*a}
    add AX, -3 {или sub ax,3}
    jmp @exit
@error: {Деление на 0}
    mov f,1 {Отметка о делении на ноль}
@exit: {Получение результата}
    mov rez, AX{Занесение
результатавпеременную} end;
writeln('Результат = ', rez);
writeln('Деление на ноль (1=Да, 0=Нет)=' , f);
End.

```

Команды управления циклами

Для организации цикла предназначены команда LOOP которая реализует классический цикл со счетчиком с постусловием.

Таблица 2.10 - Команды управления циклами

Команда	Формат	Комментарий
LOOP	LOOP метка	Повторить цикл
LOOPE/LOOPZ	LOOPE метка	Повторять, пока равны
LOOPNE/LOOPNZ	LOOPNE метка	Повторять, пока не равны

Команда LOOP служит для перехода на метку, в качестве счетчика цикла используется регистр CX. При выполнении цикла выполняется декремент содержимого регистра CX, а затем проверяет его значение, если регистр CX не равен 0, то происходит переход на метку, иначе управление переходит к следующей команде после LOOP. Содержимое CX представляет собой целочисленное число без знака. В CX нужно помещать число, равное требуемому количеству повторений цикла.

Кроме команды LOOP и команд условных переходов существуют ещё две команды, позволяющие организовывать циклы: LOOPZ (синоним LOOPE) и LOOPNZ (синоним LOOPNE). Действие этих команд напоминает LOOP, также уменьшают счетчик CX, но передают управление в начало цикла при совместном условии установки (или сброса) флага ZF и неравенства нулю счетчика CX. Переход к метке цикла осуществляется в том случае, если после декремента содержимое CX не равно 0 и выполняется условие: ZF=1 (для команды

LOOPZ/LOOPE) или ZF=0 (LOOPNZ/LOOPNE). Команды удобно использовать в алгоритмах, где цикл должен завершаться в двух случаях: выполнено требуемое количество итераций; выполнено некоторое условие досрочного завершения цикла.

Пример: Вычислить сумму 1+2+... +9+10.

```
asm
    mov AX, 0 {Накапливаем сумму в AX} mov CX, 10
    {Счетчик цикла}
    @metka: {метка обозначает начало цикла}
        add AX, CX {складываем следующее значение}
    loop @metka {переход на начало цикла при
    CX<>0}

end;
```

Строковые команды

Строковые команды служат для обработки строк. Под словом здесь понимается цепочка байт или слов длиной до 64 Кб. Команды используют примитивы: пересылка, сравнение, сканирование, загрузка и сохранение. Каждый примитив обрабатывает за один раз только один байт или одно слово.

Таблица 2.11 - Строковые команды

Команда	Формат	Комментарий
<i>Пересылка строк</i>		
MOVSB	MOVSB	Пересылать байты
MOVSW	MOVSW	Пересылать слова
<i>Сравнение строк</i>		
CMPSB	CMPSB	Сравнивать байты
CMPSW	CMPSW	Сравнивать слова
<i>Сканирование</i>		
SCASB	SCASB	Искать байт
SCASW	SCASW	Искать слово
<i>Загрузка данных</i>		
LODSB	LODSB	Загружать байты
LODSW	LODSW	Загружать слова
<i>Сохранение данных</i>		
STOSB	STOSB	Сохранять байты
STOSW	STOSW	Сохранять слова

При использовании строковых команд важно помнить, что команды всегда берут

адрес строки-источника из пары DS:SI, а строки-приемника - из пары ES:DI. Перед исполнением строковой команды необходимо инициировать сегментные регистры нужным образом. Строковые команды используют индексную адресацию с автоматическим изменением смещения в SI/DI после однократного исполнения примитива. Содержимое этих регистров изменяется на 1 при обработке байтов и на 2 при обработке слов, причем наращивается, если флаг направления DF сброшен, и уменьшается, если он равен 1.

Команды загрузки данных LODSB и LODSW загружают байт или слово из памяти в регистр аккумулятора AX. Команды STOSB и STOSW сохраняют данные из регистра аккумулятора AX в ячейку памяти. Команды MOVSB и MOVSW аналогичны командам LODSB/LODSW и STOSB/STOSW, если их использовать совместно. Содержимое регистра AX не изменяется, потому что через регистры байты или слова не передаются.

Команды пересылки строк MOVSB и MOVSW работают быстрее, чем комбинация команд LODSB/STOSB и LODSW/STOSW. Команды MOVSB и MOVSW имеют минимально возможную для инструкции длину и занимают только один байт.

Команды SCASB и SCASW используются для просмотра памяти и поиска совпадения с конкретным значением размером в байт или слово.

В программах на языке Pascal сегментный регистр DS всегда будет содержать сегмент данных, инициировать который необязательно. Регистр дополнительного сегмента ES, содержит ссылку на сегмент данных, поэтому необходимо проводить инициацию (PUSH DS; POP ES) перед тем как воспользоваться строковыми командами.

Пример: Дан массив байт. Получить другой массив, в котором каждый элемент первого массива увеличен на 3.

```
Program sample;
{Исходный массив организован в виде константы}
const j:array[0..4] of byte =(1,2,3,4,5);
var k:array[0..4] of byte; {Массив результата}
begin
asm
    mov CX, 5 {Количество элементов в массиве}
    {Загрузка смещения исходного массива j в SI}
    mov SI, offset j {или lea SI, j}
    mov AX, SEG j {Занесение значения элемента
```

```

массива j воспользовавшись префиксом SEG в AX}
    {Загрузка смещения массива результата k в DI}
    mov DI, offset k {или lea DI, j}
    mov AX, SEG k {Занесение значения массива k}
    mov ES, AX {В ES заносится значения из AX}
    @metka: {Метка обозначает начало цикла}
    lodsb {Загрузка очередного байта массива j}
    add AL, 3 {Увеличение байта на 3}
    stosb {Запись байта в массив результата k}
    loop @metka {Переход на начало цикла при CX<>0}
end;
writeln('Массив j= ', j[0], j[1], j[2], j[3], j[4]);
writeln('Массив k= ', k[0], k[1], k[2], k[3], k[4]);
end.

```

Команды прерываний

Прерывание - специализированная программа, которая находится постоянно во время сеанса в операционной системе и используется в процессе обмена микропроцессора и периферийного устройства или исключительной ситуации при работе микропроцессора.

Таблица 2.12 - Команды прерываний

Команда	Формат	Комментарий
INT	INT <u>номер</u>	Выполнить прерывание
INTO	INTO	Выполнить прерывание по переполнению
IRET	IRET	Вернуться из прерывания

Команды INT и INTO служат для организации прерывания, которое выполняется, если флаг переполнения OF установлен. По команде организации прерывания в стек помещается регистр флагов, сегмент CS и указатель IP, а новые значения этих регистров берутся из 4-байтного вектора прерывания, соответствующего номеру прерывания в команде INT, или из вектора 4 для команды INTO. Микропроцессор сбрасывает флаги трассировки TF и прерываний IF перед передачей управления программе обработки прерывания. Для штатной работы отладчиков необходимо сбросить флаг TF, который используется прерыванием по вектору 1 либо 4, сброс флага IF блокирует вмешательство других процессов в ход обработки прерывания.

Команда IRET противоположна командам INT и INTO, она предназначена для выход из программы обработки прерывания. Команда IRET

считывает из стека 3 двухбайтных слова и помещает их в регистры IP, CS, а также в и регистр флагов.

Команды управления

Предназначены для управления флагами и внешней синхронизацией.

Таблица 2.13 - Команды управления

Команда	Формат	Комментарий
<i>Управление флагами</i>		
STC	STC	Установить перенос
CLC	CLC	Очистить перенос
CMC	CMC	Инвертировать CF
STD	STD	Установить направление
CLD	CLD	Очистить направление
STI	STI	Разрешить прерывания
CLI	CLI	Запретить прерывания
<i>Внешняя синхронизация</i>		
HLT	HLT	Остановить вычисления
WAIT	WAIT	Ждать активности на шине
ESC	ESC <u>код, источник</u>	Передать команду
LOCK	LOCK	Захватить шину
<i>Пустая команда</i>		
NOP	NOP	Нет операции

Команда внешней синхронизации HLT служит для перевода микропроцессора в состояние остановки вычислений, из этого состояния можно вывести микропроцессор только при перезагрузке или при наступлении немаскируемого прерывания.

Команда WAIT служит для ожидания сигнала обслуживания прерывания высокоприоритетного устройства типа контроллера прямого доступа к памяти. Команда WAIT заставляет микропроцессор выполнять холостой режим работы и каждые 5 тактов проверять уровень сигнала на входной шине: пока на этой шине нет сигнала активности, процессор выполняет WAIT, но как только шина активизируется, он продолжит исполнение программы.

Команда ESC управляет работой математического сопроцессора, перенаправляя указанный в ней операнда в шину данных и обеспечивая передачи команд другим процессорам. Команда имеет два операнда код - код команды сопроцессора и источник - используемый в этой команде операнд.

Команда LOCK служит для монополизации шины данных любым другим

внешним устройством (процессором), путем отправки сигнала по шине. Команда представляет собой однобайтового префикса, который можно использовать совместно с любой другой командой микропроцессора.

Ассемблерные подпрограммы

Ассемблерные подпрограммы — процедуры и функции, объявленные с директивой `Assembler`. Тело процедур и функций состоит из ассемблерных команд которые объединены ассемблерными операторными скобками `asm...end`. При компиляции таких специализированных процедур и функций параметры строкового типа длиной до 4 байт не помещаются во временную память, а находятся внутри подпрограммы как переменные. Компилятор не будет создавать переменную `@Result` для результата функции, поэтому ссылка на эту переменную в ассемблерной функции недопустима, сослаться на переменную `@Result` можно, только если функция возвращает значения строкового типа.

Пример: Ассемблерная подпрограмма: процедура сравнения двух чисел

```
Procedure sample2(x,y:word); Assembler;
asm
    mov AX, a {Занесение в AX переменнойa}
    mov BX, b {Занесение в AX переменнойb}
    cmp AX, BX {Сравнение a и b}
    sub AX, 5 {Если a,b равны}
end;
```

Пример: Ассемблерная подпрограмма: функция сложения двух чисел

```
Function sample1(X,Y:Integer):LongInt; Assembler;
asm
    mov AX, X {Занесение в AX переменной X}
    mov BX, Y {Занесение в BX переменной Y}
    add AX, BX {AX = AX + BX}
end;
```

Ассемблерные функции должны возвращать результат в виде переменных:

- длиной 1 байт (`Byte`, `Char` и т.п.) - в регистре `AL`;
- длиной 2 байта (`Integer`, `Word`) - в регистре `AX`;
- длиной 4 байта (`Pointer`, `LongInt`) - в регистрах `DX` (старшее слово) и `AX` (младшее слово);
- типа `Real` - в регистрах `DX`, `BX`, `AX` (старшее слово в `DX`, младшее в `AX`);
- вещественных типов `Single`, `Double`, `Extended`, `Comp` - в

регистре ST(0) сопроцессора;

– строкового типа - во временной области памяти, на которую ссылается @Result.

Способы адресации в языке Ассемблер

При изучении команды следует рассмотреть и способы адресации в Ассемблере. В архитектуре микропроцессоров i8086/8088 адрес любого байта задается двумя 16-битовыми словами: сегментом и смещением. При формировании 20-битного адреса, который необходим для адресации в пределах 1 Мб, сегмент сдвигается влево на 4 разряда, т.е. умножается на 16 и складывается со смещением. Емкость 16-битного смещения составляет 65536 значений, в пределах одного сегмента можно адресовать до 64 Кб. Архитектура микропроцессоров позволяет использовать семь способов адресации:

Регистровая адресация

При регистровой адресации происходит изъятие операнда из регистра или наоборот помещение его в регистр. Операнды могут располагаться как в сегментных регистрах, так и в регистрах общего назначения.

Примеры:

```
mov AX, BX {Извлечение из BX и помещаем в AX}
add CX, AX {К CX добавляется содержимое AX}
push BX {Занесение в стек содержимого BX}
```

Непосредственная адресация

Все арифметические команды, кроме деления позволяют указывать в тексте программы один из операндов в виде 8- или 16-битной константы.

Примеры:

```
mov AX, 55 {Загрузка в AX значение 100}
add DX, 5 {К регистру DX прибавляется 5}
mov BX, $FFE3 {Занесение в CX значения 65507}
```

Прямая адресация (адресация по смещению)

Адрес операнда располагается в памяти и является словом находящемся в сегменте, на который указывает ES, со смещением от начала сегмента 0001.

Пример:

```
mov AX, ES:[0001h] {Помещение слова в регистр AX}
```

Косвенная регистровая

Для организации косвенной адресации необходимо смещение хранить в

регистрах BX, BP, SI или DI и заключать эти регистры в квадратные скобки. Каждый из таких регистров по умолчанию работает со своим сегментным регистром: DS:BX, SS:BP, DS:SI, ES:DI.

Пример: Переместить содержимое 16-битного слова из памяти с адресом DS:BX в регистр AX `mov AX, [BX]`

Возможно, указать сегментный регистр, если он отличается от DS:BX, SS:BP, DS:SI, ES:DI.

Пример: `mov AX, ES:[BX]`

Адресация по базе

Регистры BX и BP содержат адрес начала фрагмента памяти т.е. базы, относительно которой ассемблер вычисляет смещение.

Пример: Загрузить в AX 5-ый по счету байт от начала базы памяти по адресу DS:BX `mov AX, [BX]+5`

Индексная адресация

Содержимое индексных регистров SI и DI указывают положение элемента относительно начала некоторой области памяти.

Пример: Переслать 16 элемент массива ABC типа Byte в регистр AH.

```
mov SI, 15 {Занесение в SI константу 15}
```

```
mov AH, ABC[SI] {Пересылка в AH 16-й по порядку  
байт от начала массива}
```

Адресация по базе с индексированием

Вариант индексной адресации для случая, когда индексруемая область памяти задается своей базой.

Пример: `mov AX, [BX][SI]`

2.2 Организация прерываний

Прерывание - специальная программа находящаяся постоянно во время сеанса в операционной системе и используется в процессе обмена микропроцессора и периферийных устройств или исключительной ситуации при работе микропроцессора

Прерывания VESA BIOS

Базовая система ввода-вывода или basic input/output system (BIOS) - служит для выполнения обработки прерываний с номерами от 00h до 1Fh. Такие прерывания имеют отношения к работе устройств компьютера: монитор, клавиатура, принтер и др.

К прерываниям BIOS относятся:

- 00h: Деление на ноль.
- 01h: Пошаговое прерывание.
- 02h: Немаскируемое прерывание.
- 03h: Точка прерывания.
- 04h: Переполнение.
- 05h: Печать экрана.
- 06h и 07h: Зарезервированы.
- 08h: Таймер.
- 09h: Клавиатура.
- 0Ah-0Dh: Работа с портами
- 0Eh: Дискета.
- 0Fh: Работа с принтером.
- 10h: Видео сервис. Работа с видеоадаптерами (видеоконтроллерами).
- 11h: Список оборудования.
- 12h: Размер используемой памяти.
- 13h: Дисковый в/в.
- 14h: В/в через последовательный порт
- 15h: Расширенный сервис АТ.
- 16h: В/в клавиатуры.
- 17h: В/в принтера.
- 18h: ROM-BASIC.
- 19h: Загрузка.
- 1Ah: В/в таймера.
- 1Bh: Прерывание клавиатуры.
- 1Ch: Пользовательское прерывание по таймеру
- 1Dh: Видео параметры
- 1Eh: Параметры дискет.
- 1Fh: Символы графики.

Стандарт VESA (Video Electronics Standards Association) был разработан для унификации операций (установка видеорежимов и управления видеопамью) при работе с видеоадаптерами SVGA. Стандартный видеоадаптер

VGA имеет 256 Кб видеопамати, а для SVGA имеет от 512 Кб до 32 Мб видеопамати. Расширение BIOS для стандарта VESA называется VESA BIOS Extensions (VBE)

Видеорежимы видеоадаптеров:

- VGA (Video Graphics Array)

Таблица 2.14 - Видеорежимы VGA

Номер видеорежима	Разрешение	Количество цветов	Объем памяти
11h	640x480	2	64 Кб
12h	640x480	16	64 Кб
13h	320x200	256	64 Кб

- SVGA (Super Video Graphics Array)

Таблица 2.15 - Видеорежимы SVGA (VESA)

Номер видеорежима	Разрешение	Количество цветов	Объем памяти
100h	640x400	256	256 Кб
101h	640x480	256	320 Кб
102h	800x600	16	256 Кб
103h	800x600	256	512 Кб
104h	1024x768	16	192 Кб
105h	1024x768	256	768 Кб
106h	1280x1024	16	768 Кб
107h	1280x1024	256	1,3 Мб
10Dh	320x200	32768 (32К)	128 Кб
10Eh	320x200	65536 (64К)	128 Кб
10Fh	320x200	16777216 (16М)	192 Кб
110h	640x480	32768 (32К)	768 Кб
111h	640x480	65536 (64К)	768 Кб
112h	640x480	16777216 (16М)	1 Мб
113h	800x600	32768 (32К)	1 Мб
114h	800x600	65536 (64К)	1 Мб
115h	800x600	16777216 (16М)	1,4 Мб
116h	1024x768	32768 (32К)	1,5 Мб
117h	1024x768	65536 (64К)	1,5 Мб
118h	1024x768	16777216 (16М)	2,3 Мб
119h	1280x1024	32768 (32К)	2,5 Мб
11Ah	1280x1024	65536 (64К)	2,5 Мб
11Bh	1280x1024	16777216 (16М)	3,7 Мб

Таблица 2.16 - Текстовые режимы

Номер видеорежима	Разрешение
<i>Стандартные текстовые режимы</i>	
02h	40x25
03h	80x25
<i>Текстовые режимы VESA</i>	
108h	80x60
109h	132x25
10Ah	132x43
10Bh	132x50
10Ch	132x60

Рассмотрим наиболее полезные функции VESA, которые поддерживаются распространенными в настоящее время моделями видеоконтроллеров.

Выбор видеорежима

BIOS позволяет переключать экран монитора в различные режимы как текстовые, так и графические. Режимы отличаются различным разрешением и количеством строк и столбцов, а также количеством цветов.

Прерывание INT 10h, Функция AH = 00 «Установить видеорежим»

Ввод: AL = номер режима в младших 7 битах

Вывод: Ничего не возвращает.

Вызов функции AH = 00 переводит экран в выбранный режим. Если старший бит AL не установлен в 1, экран очищается. В регистр AL можно установить VGA режимы: видеорежим и стандартный текстовый режим. Если видеоадаптер поддерживает стандарт VESA BIOS, в режимы с высоким разрешением можно переключаться, используя функцию 4Fh.

Прерывание INT 10h, Функция AH = 4Fh, Подфункция AL = 02 «Установить SVGA-видеорежим»

Функция 4Fh устанавливает видеорежим с заданным номером и режим адресации видеопамяти.

Ввод: BX = номер видеорежима или текстового режима VESA

Вывод: AL = 4Fh, если эта функция поддерживается

АН = 0, если переключение произошло успешно

АН = 1, если произошла ошибка

АН = 2, если функция не поддерживается в данной аппаратурой

Режимы с номерами меньше 100h определяются изготовителем видеоконтроллера и не будут обрабатываться функциями VESA. Если функции установки видеорежима будет послан код со значением меньше 100h, то она передаст его на обработку прерыванию BIOS 10h.

Работа с видеопамятью

Для отображения видеопамати на основное адресное пространство используется 64000 байт и располагается с адреса A000h:0000h. Увеличение разрешения или числа цветов приводит к увеличению объема видеопамати и превышает максимальные границы сегмента в реальном режиме равном 65535 байт, а затем и размер участка адресного пространства, которое отводится для видеопамати 160 Кб, от A000h:0000h до B800h:FFFFh. Область памяти BIOS начинается с адреса C800h:0000h.

Чтобы вывести изображение, используются два механизма переключение банков видеопамати:

- Линейный кадровый буфер для защищенного режима (Linear Frame Buffer)

Видеопамать находится в непрерывном массиве адресного пространства, начинающегося с любого адреса, но не с адреса A000h. Это обусловлено тем, чтобы вся видеопамать, которая занимает несколько Мб, поместилась в один непрерывный массив. В защищенном режиме максимальный размер сегмента составляет 4 Гб. Линейный кадровый буфер можно использовать, только если видеоадаптер поддерживает спецификацию VESA BIOS Extensions 2 (УВЕ 2).

- Реальный режим

В реальном режиме вывод на экран изображения осуществляется копированием данных в сегмент равный 64 Кб, с адреса A000h:0000h, но эта область памяти соответствует только части экрана. Для вывода изображения в

другую часть экрана, необходимо вызвать функцию переключить банк видеопамати (4F05h), которая изменит область видеопамати, которой соответствует сегмент A000h.

Пример: В режиме 640x480 с 256 цветами требуется 320 Кб для хранения всего видеоизображения. Заполнение сегмента A000h:0000h - A000h:FFFFh приводит к закраске 0,2 части экрана, переключение на один банк памяти и повторное заполнение этой же области приводит к закраске следующей 0,2 части экрана. Перемещение окна осуществляется двумя способами: первый способ - передача управления прямо в процедуру, предварительно вызвав подфункцию 01 видеофункции 4Fh которая вернет адрес этой процедуры; второй способ - воспользоваться подфункцией 05 видеофункции 4Fh.

Прерывание INT 10h, Функция AH = 4Fh, Подфункция AL = 05
«Управление окнами видеопамати».

Функция предназначена для управления отображением видеопамати на адресное пространство процессора при использовании страничного доступа через окно 64 Кб.

Ввод: AX = код 4F05h;

BH = код выполняемой операции:

0 - установить окно видеопамати,

1 - получить номер текущего окна видеопамати)

BL = номер окна (0 - окно А, 1 - окно В);

DX = номер окна в видеопамати (задается только при BH = 0).

Вывод: AX = код возврата;

DX = адрес окна в видеопамати (выдается только при BH = 1).

Подфункция 05h применяется в оконном режиме адресации, когда в каждый момент времени процессору доступен только один участок (сегмент) видеопамати размером в 64 Кб. Если длина строки в пикселах не кратна 2^N , то сегмент будет содержать нецелое число экранных строк, и границы сегментов не будут совпадать с границами экрана ([рисунок 2.23](#)).



Рисунок 2.23 - Организация видеопамати в режиме страничной адресации

В результате при выводе изображения на экран необходимо контролировать все границы сегмента, так как вывод выполняется по точкам, слева направо и сверху вниз. Вывод точки на экран выполняется командой пересылки данных MOV. Подфункцию 05h необходимо вызывать каждый раз, когда требуется перейти к работе с другим сегментом. Операцию контроля необходимо включать внутри всех циклов вывода изображения, что сильно замедляет работу, особенно на современных видеоконтроллерах, у которых скорость записи в память измеряется десятками мегабайт в секунду.

Прерывание INT10h, Функция AH = 4Fh, Подфункция AL = 06 «Получить или установить длину логической строки развертки».

Функция читает текущую длину логической строки и устанавливает ее новое значение.

Ввод: AX = код 4F0 6h;

BL = код выполняемой операции:

- 1 - установить длину строки развертки в пикселях;
- 2 - получить текущую длину строки развертки в пикселях;
- 3 - установить длину строки развертки в байтах;
- 4 - получить макс. возможную длину строки развертки в пикселях;

CX = ширина строки (при BL = 0 - в пикселях, при BL = 2 - в байтах).

Вывод: AX = код возврата;

VX = число байт на строку развертки;

CX = число пикселей на строку развертки (округление до целого

пикселя);

DX = максимальное число строк развертки.

Подфункция $06h$ служит для реорганизации видеопамати с целью упрощения адресации. Реорганизация производится путем выравнивания длины строки на величину, равную $2N - 1024$ или 2048 . После выравнивания длины строки в пикселях на $2N$, сегмент будет состоять из целого числа экранных строк, а начало каждого сегмента будет совпадать с началом строки экрана ([рисунок 2.24](#)).

Сегмент №1
Сегмент №2
Сегмент №3
Сегмент №4
Сегмент №5
Сегмент №6

Рисунок 2.24 - Видеопамать после реорганизации

При выравнивании логической длины строки на $2N$ используются сдвиги и логические операции, что значительно упрощает вычисление адресов памяти. После реорганизации становится возможным реализовать плавную прокрутку изображения с использованием аппаратных средств видеоадаптера.

Видеоадаптеры позволяют делать длину строки в видеопамати больше физической длины строки экрана за счет реорганизации неиспользуемых областей памяти.

При реорганизации операция контроля пересечения границы сегмента выполняется во внешнем цикле вывода строки изображения, а не во внутреннем цикле рисования точек. Объем памяти при низких разрешениях не является препятствием для операции выравнивания даже для видеоадаптеров, не обладающих высокой производительностью.

Установка графической точки

Для установки графической точки на экране служит функция $0Ch$, но работает крайне медленно и не используется при разработке сложных программ.

Прерывание $INT 10h$, Функция $AH = 0Ch$ «Вывести точку на экран».

Ввод: AH = код $0Ch$; AL = номер цвета.

BL = номер видеостраницы;

DX = номер строки;

CX = номер столбца;

Вывод: Ничего не возвращает.

Для разработки сложных программ требующих вывода на экран динамической графической информации программируют на уровне портов ввода-вывода видеоконтроллера.

Программирование на уровне портов ввода-вывода

Многие возможности компьютера могут быть реализованы только при программировании на уровне портов ввода-вывода. Порт - специальный адрес, по которому хранится, входной и выходной регистр периферийного устройства.

Видеоадаптерами можно управлять с помощью портов ввода-вывода ([таблица 2.17](#)) и внутренними регистрами к которым можно обратиться.

Таблица 2.17 - Порты ввода-вывода видеоадаптеров

Номер порта	Тип порта	Назначение
03CCh, 03C2h, 03CAh	индексный	Определяют состоянием видеоадаптера и обеспечивают корректный доступ к регистрам
03DAh, 03C2h, 03DAh	данных	
03C4h	индексный	Управляют временными параметрами видеоадаптера и разрешением/запрещением доступа к отдельным цветовым плоскостям.
03C5h	данных	
03D4h	индексный	Управляют сигналами синхронизации, формой курсора и форматом данных на экране.
03D5h	данных	
03CEh	индексный	Управляет обменом данными между процессором и видеопамятью.
03CFh	данных	
03C0h, 03C6h-03C9h	индексный и данных	Управление цветовыми характеристиками изображений и палитрой цветов.

Для программирования различных режимов записи графических данных, чтения и перемещения данных используются порты ввода-вывода 03CEh и 03CFh. Для обращения к регистрам видеоадаптера следует передать в порт 03CEh индекс нужного регистра, а в порт 03CFh передать данные для выбранного регистра видеоадаптера. Для записи данных в регистр видеоадаптера, нужно поместить индекс в AL, а посылаемый байт в AH и выполнить команду вывода слова в порт 03CEh.

Рассмотрим несколько регистров видеоадаптера:

- 00h: Регистр установки/сброса.

Биты 0-3: установка/сброс цветовой плоскости.

- 01h: Регистр разрешения установки/сброса.

Биты 3-0: включают режим установки/сброса для цветовой плоскости. Данные для одних цветовых слоев получают от CPU, а для других — из регистра установки/сброса. Режим действует только в нулевом режиме работы (регистр 05h)

- 02h: Регистр сравнения цвета.

Биты 3-0: искомые биты для цветowych плоскостей. Регистр используется для поиска пикселя заданного цвета, чтобы не обращаться по очереди во все цветowych слои.

- 03h: Регистр циклического сдвига данных.

Биты 4-3: выбор логической операции:

- 00 - данные от CPU записываются без изменений
- 01 - операция AND над CPU и регистром-защелкой
- 10 - операция OR над CPU и регистром-защелкой
- 11 - операция XOR над CPU и регистром-защелкой.

Биты 2-0: на сколько бит выполнять вправо циклический сдвиг данных перед записью в видеопамять.

- 04h: Регистр выбора читаемой плоскости.

Биты 1-0: номер плоскости (0-3). Запись сюда изменяет номер цветовой плоскости, данные из которой получает CPU при чтении из видеопамяти.

- 05h: Регистр выбора режима работы.

Бит 6: 1/0— 256/16 цветов,

Бит 4: четные адреса - плоскости 0, 2, нечетные — 1,3

Бит 3: 1 — режим сравнения цветов

Биты 1-0: режим:

- 00 - данные из CPU (бит на пиксель) + установка/сброс + циклический сдвиг+ логические функции
- 01 - данные в/из регистра-защелки - прочитать его и записать в другую область памяти.
- 10 - данные из CPU, байт на пиксель, младшие 4 бита записываются в соответствующие плоскости
- 11 - данные из CPU, байт на пиксель, младшие 4 бита записываются в соответствующие плоскости + режим битовой маски.

- 06h: Многоцелевой регистр графического контроллера.

Бит 3 - 2: видеопамять:

00 - A0000h – BFFFFh (128 Кб)

01 - A0000h – AFFFFh (64 Кб)

10 - B0000h – B7FFFh (32 Кб)

11 - B8000h – BFFFFh (32 Кб)

Бит 0: 1/0 - графический/текстовый режим.

- 07h: Регистр игнорирования цветowych плоскостей.

Биты 3 - 0 игнорировать цветовую плоскость (0 - 3).

- 08h: Регистр битовой маски.

Если бит 0, то бит будет браться из регистра-защелки, а не от CPU.

Чтобы занести данные в регистр-защелку, надо выполнить одну операцию чтения из видеопамати, при этом в каждый из четырех регистров-защелок будет помещено по одному байту из соответствующей цветовой плоскости.

Использования прерываний в программах

Для использования прерываний в программах, написанных на языке Pascal, можно воспользоваться одним из двух способов.

- При помощи специального типа данных Registers, предназначенного для обращения к регистрам процессора, и процедуры Intr, вызывающей прерывание с указанным номером. Как тип данных Registers, так и процедура Intr реализованы в стандартном библиотечном модуле DOS.

- При помощи команд встроенного ассемблера. Для вызова прерывания используется операция INT.

Библиотека DOS

Библиотека DOS предназначена для работы системными средствами операционной системы DOS версии 3.0 и выше. При использовании любых библиотек Turbo Pascal запрещено переопределять типы и использовать имена, совпадающие с именами или константами, входящими в библиотеку. [15]

Все процедуры и функции данной группы работают с типом Register.

```
Type Registers = record
```

```
Case Integer of
```

```
0: (AX, BX, CX, DX, BP, SI, DI, DS, ES, flags: word);
```

```
1: (AL, AH, BL, BH, CL, CH, DL, DH: byte) End;
```

В библиотеке DOS доступны только те регистры, которые определены первой частью вариантной записи типа Register.

- Выполнить прерывание:

```
Intr(IntNo: byte, Var R: Registers);
```

Где IntNo - № прерывания, а R - содержимое регистров.

- Выполнить прерывания MS-DOS:
MSDOS (Var R:Registers);
- Сохранить вектора прерываний:
SwapVectors

При употреблении прерывания с использованием библиотеки DOS рекомендуется перед и после прерывания вызывать процедуру SwapVectors, так как Turbo Pascal имеет собственную программу по обработке прерываний от данных устройств.

Установка видеорежима

Рассмотрим на примере установку видеорежима для VGA и SVGA

Пример: Установки VGA видеорежима 320x200 и 256 цветов Procedure

```
InitGraphDos;
var R: Registers; {Переменная типа Registers} begin
  SwapVectors; {Сохранение вектора прерываний}
  R.AH:=$00; {Функция установки видеорежима}
  R.AL:=$13; {Видеорежим (табл. 2.14 и 2.16)}
  Intr($10,R); {Выполнение прерывания}
  SwapVectors; {Сохранение вектора прерываний}
end;
```

Пример: Установки SVGA видеорежима 1024x768 и 16 цветов.

```
Procedure InitGraphVESADos;
var R: Registers; {Переменная типа Registers}
begin
  SwapVectors; {Сохранение вектора прерываний}
  R.AH:=$4F; {Функция работы с SVGA (VESA)}
  R.AL:=$02; {Подфункция установки видеорежима}
  {или занести в регистр AX, R.AX:=$4F02;}
  R.BX:=$104; {Видеорежим (табл. 2.14 и 2.16)}
  Intr($10,R); {Выполнение прерывания}
  SwapVectors; {Сохранение вектора прерываний}
end;
```

Установка графической точки

Пример: Установка графической точки с использованием функции OSH

```
Procedure SetPixelDos(x,y: word; color: byte);
var R: Registers; {Переменная типа registers}
begin
  SwapVectors; {Сохранение вектора прерываний}
```

```

R.AH:=$0C; {Функция установки точки на экран}
R.AL:=color; {Задание цвета точки}
R.BH:= 00; {Номер видеостраницы}
R.CH:=x; {Номер столбца}
R.DX:=y; {Номер строки}
Intr($10,R); {Выполнение прерывания}
SwapVectors; {Сохранение вектора прерываний}
end;

```

Пример: Установка графической точки на уровне портов ввода-вывода. Вывод точки на экран 800x600 16 цветов (102h). 1 бит отображаемой памяти равен одному пикселю изображения.

Синхронный ввод и вывод информации осуществляется, используя predetermined массив Port. Массив используются для написания системных программ без обращения к драйверам операционной системы. После изменения содержимого порта, информация должна быть в нем восстановлена.

Port [<Addr>] - осуществляет связь с портом. Порт имеет абсолютно точный адрес, состоящий из трех шестнадцатеричных цифр ([таблица 2.17](#)).

Пример: Установка графической точки с использованием портов ввода-вывода видеоадаптера

```

{Процедура записи в порт}
Procedure PortIO(register, value: Integer);
begin
    Port[$3CE]:=register; {Индексный порт}
    Port[$3CF]:=value; {Порт данных}
end;

{Процедура установки пикселя}
Procedure SetPixelDos(x, y : Word; color : Byte);
var Sm:Integer; b,bb,p:Byte;
begin
    b:=x mod 8; {Номер пикселя в байте}
    b:=128 shr (b); {Маска для битов в байте}
    PortIO(8,b); {Регистр битовой маски}
    PortIO(5,0); {Регистр режима записи, режим 0}
    PortIO(1,$0F);{Регистр разрешения уст./сброс}
    PortIO(0,color);{установка цвета, регистр
                    установка/сброс}

```

```

Sm:=(y*100+x div 8);{Выбор необходимого байт в
отображаемой памяти}
bb:=Mem[$A000:Sm];{Чтение байта}
Mem[$A000:Sm]:=$FF;{Запись байта}
end;

```

Встроенный ассемблер

Для выполнения прерывания в ассемблере используется процедура
Int (Номер прерывания).

Установка видеорежима

Пример установки VGA видеорежима 320x200 и 256 цветов Procedure

```

InitGraphAsm; assembler; asm
    mov AH,00h {Функция установки видеорежима }
    mov AL,13h {Видеорежим (табл. 2.14 и 2.16)}
    int 10h {Выполнение прерывания }
end;

```

*Пример: Установки SVGA видеорежима с проверкой возможности его
установления. Номер видеорежима устанавливается переменной mode.*

```

Function InitGraphAsm (m: word): boolean;
assembler; asm
    cmp m,100h {Проверка выбранного режима}
    jb @Normal_VGA {Если номер видеорежима ниже
                    100h то выполняется в
                    стандартом VGA режиме, если
                    больше то SVGA VESA}
    mov AX, 4F02h {Функция работы с SVGA (VESA)
                  и вызов Подфункция установки
                  видеорежима }
    mov BX, m {Видеорежим (табл. 2.14 и 2.16)}
    int 10h {Выполнение прерывания}
    cmp AX, 004Fh {VESA поддерживается}
    jne @Error {если не поддерживается}
    mov AL, true jmp @done @Error:
        mov AL, false jmp @done
@Normal_VGA:
    mov AX, mode {AH будет равна нулю}
    int 10h

```



```
        mov AL, true
@done: end;
```

Установка графической точки

Пример установки графической точки с использованием функции 0CH

```
Procedure SetPixelAsm(x,y:word; c:byte);
assembler; asm
        mov AH, 0CH {Функция установки точки на экран}
        mov AL, c {Задание цвета точки}
        mov CX, x {Номер столбца}
        mov DX, y {Номер строки}
        int 10h {Выполнение прерывания}
end;
```

Пример: Установка графической точки на уровне портов ввода-вывода

```

Procedure PutPixel(x,y:word; c: byte); assembler;
asm
    push DS
    mov BX, [x]
    mov CX, BX
    and CL, 7
    mov CH, 10000000b
    shr CH, CL {CH = битовая маска}
    mov DX, 80
    mov AX, [y] {Y координата}
    mul DX {AX = Y * 80}
    mov CL, 3
    shr BX, CL {BX = X / 8}
    add BX, AX {BX = Y * 80 + X / 8}
    mov AX, 0A000h
    mov DS, AX {DS:BX -> пиксель в буфер}
    {Установка данных в видеоконтроллер}
    mov DX, 3CFH
    mov AX, 0005
    out DX, AX {установка значения 0}
    mov AH, [c] {установка цвета}
    mov AL, 0
    out DX, AX {перезапуск регистра}
    mov AX, 0F01h
    out DX, AX {перезапуск регистра}
    mov AH, CH
    mov AL, 8
    out DX, AX {обновление пикселя}
    or [BX], AL {обновление}
    {обнуление значений параметров видеоадаптера}
    mov AX, 0000
    out DX, AX {значение по умолчанию}
    mov AX, 0001
    out DX, AX
    mov AX, 0FF08h
    out DX, AX {значение по умолчанию битовой маски}
    pop DS
end;

```

3 ПРАВИЛА ОФОРМЛЕНИЯ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

3.1 Общие правила

Пояснительная записка должна быть напечатана на одной стороне листа бумаги формата А4, кегль - 14, шрифт - "Times New Roman", интервал полуторный, форматирование по ширине страницы. Переносы в пояснительной записке не допускаются. Размеры полей: левое - 30 мм, правое - 15 мм, верхнее и нижнее поля - 20 мм каждое.

Расстояние между заголовком и последующим текстом не менее 6 пт; между заголовком раздела и подраздела - 3 пт. Интервал между текстом предыдущего раздела или подраздела и заголовком последующего должен быть не меньше 6 пт. Абзацы в тексте начинаются отступом, равным 12,5 мм. Разделы и подразделы должны иметь порядковые номера, обозначенные арабскими цифрами без точки и записанные с абзацного отступа.

Заголовки глав, пунктов и подпунктов должны быть краткими и должны точно отражать содержание раздела. Заголовки глав, пунктов и подпунктов записывают в виде предложения с абзацного отступа, с прописной буквы, без точки в конце. Переносы слов в заголовках не допускаются. Точка в конце заголовков не ставится. Главы записки рекомендуется начинать с нового листа.

Нумерация страниц начинается с титульного листа и заканчивается последним листом пояснительной записки. Номер страницы проставляется в центре нижней части листа без точки. На титульном листе номер страницы не указывается.

Оформление содержания

Содержание включает все пронумерованные заголовки. Наименования, включенные в содержание, записываются, начиная с прописной буквы. Для каждого названия указывается номер страницы, на которой оно находится. Не следует использовать в содержании сокращения, кроме общепринятых обозначений.

Нумерация разделов записки

Главы нумеруются в пределах пояснительной записки порядковыми номерами арабскими цифрами. Пункты нумеруются по порядку в пределах главы, а подпункты нумеруются в пределах пункта. Введение, заключение и список использованных источников не нумеруются.

Пример:

4.2 - (пункт 2 главы 4).

2.1.1 - (подпункт 1 пункта 1 главы 2).

3.2 Правила оформления графического материала

Пояснительная записка должна быть напечатана на одной стороне листа бумаги формата А4, кегль – 14, шрифт – "Times New Roman", интервал полуторный, форматирование по ширине страницы. Переносы в пояснительной записке не допускаются. Размеры полей: левое – 30 мм, правое – 15 мм, верхнее и нижнее поля – 20 мм каждое. Расстояние между заголовком и последующим текстом не менее 6 пт; между заголовком раздела и подраздела – 3 пт. Интервал между текстом предыдущего раздела или подраздела и заголовком последующего должен быть не меньше 6 пт. Абзацы в тексте начинаются отступом, равным 1.25 см. Разделы и подразделы должны иметь порядковые номера, обозначенные арабскими цифрами без точки и записанные с абзацного отступа. Заголовки глав, пунктов и подпунктов должны быть краткими и должны точно отражать содержание раздела. Заголовки глав, пунктов и подпунктов записывают в виде предложения с абзацного отступа, с прописной буквы, без точки в конце. Переносы слов в заголовках не допускаются. Точка в конце заголовков не ставится. Главы записки рекомендуется начинать с нового листа. Нумерация страниц начинается с титульного листа и заканчивается последним листом пояснительной записки. Номер страницы проставляется в центре нижней части листа без точки. На титульном листе номер страницы не указывается.

Оформление содержания

Содержание включает все пронумерованные заголовки. Наименования, включенные в содержание, записываются, начиная с прописной буквы. Для каждого названия указывается номер страницы, на которой оно находится. Не следует использовать в содержании сокращения, кроме общепринятых обозначений.

Нумерация разделов записки

Главы нумеруются в пределах пояснительной записки порядковыми номерами арабскими цифрами. Пункты нумеруются по порядку в пределах главы, а подпункты нумеруются в пределах пункта. Введение, заключение и

список использованных источников не нумеруются. *Пример:* 4.2 – (пункт 2 главы 4). 2.1.1 – (подпункт 1 пункта 1 главы 2).

Оформление рисунков

Иллюстрации могут быть выполнены как черно-белом, так и в цветном исполнении. Рисунки нумеруются в пределах главы. *Пример:* Рисунок 3.1 – (рисунок первый в третьей главе). Рисунок может содержать: название, поясняющие надписи, если они необходимы, расположенные под рисунком и номер рисунка под поясняющей надписью.

Номер рисунка и его название располагаются под рисунком или под пояснительными надписями посередине строки. Слово «Рисунок» записывается полностью, с прописной буквы без точки после номера. Название рисунка записывается в продолжении строки через дефис с прописной буквы. Пример оформления приведен на рисунке 3.1.

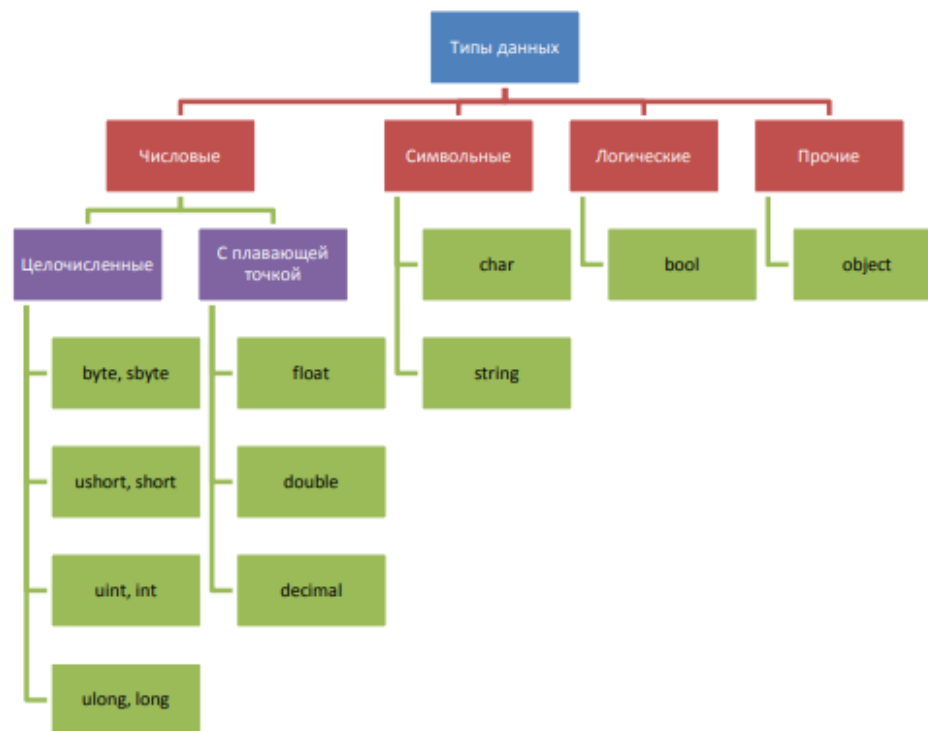


Рисунок 3.1 – Структура типов данных

Если рисунок располагается на нескольких листах, то на каждом последующем листе указывается номер рисунка, за которым следует слово

"Продолжение". *Пример:* Рисунок 3.1 Продолжение

При ссылках на иллюстрации следует писать «...в соответствии с рисунком 1» (при сквозной нумерации) или «... в соответствии с рисунком 4.1» (при нумерации в пределах раздела). Сокращения слова «рисунок» не допускается. Иллюстрации каждого приложения обозначают отдельной нумерацией арабскими цифрами с добавлением перед цифрой обозначения приложения.

Пример: Рисунок А.3 – Блок-схема

Оформление таблиц

Таблицы нумеруются арабскими цифрами. Например, «Таблица 1» или «Таблица 3.1» (первая таблица в разделе (главе) 3). Таблицы приложения обозначаются аналогично, но с добавлением перед цифрой обозначения приложения. *Пример:* Таблица Б.1 – (таблица 1 приложения Б).

Таблица может иметь заголовок, который размещается над таблицей слева без абзацного отступа в одну строку с ее номером через тире и записывается строчными буквами (кроме первой прописной). Таблицы выравниваются по ширине окна. Общий вид таблицы рассмотрим на примере таблицы 3.1.

Таблица 3.1 – Общий вид таблицы

Таблицу помещается под текстом, в котором впервые дана ссылка на неё. Допускается применять размер шрифта в таблице меньший, чем в тексте, но не менее 8.

Таблица может располагаться на нескольких листах. Заголовок таблицы указывается только на первом листе и нижняя граница таблицы не проводится. На последующих листах кроме последнего в правом верхнем углу пишется "Продолжение таблицы ..." и указывается номер. Точка после номера не ставится.

Пример: Продолжение таблицы 3.1

Продолжение таблицы 3.1

На последнем листе, где заканчивается таблица в правом верхнем углу пишется "Окончание таблицы ..." и указывается номер. Например: Окончание таблицы 3.1

Окончание таблицы 3.1

Оформление приложений

Иллюстрации, таблицы или тексты вспомогательного характера допускается оформлять в виде приложений. На все приложения в тексте работы должны быть даны ссылки. Приложения располагаются в порядке ссылок на них в тексте. Каждое приложение начинается с новой страницы, с указанием наверху посередине страницы слова «Приложение», его обозначения. Приложение должно иметь заголовок, который записывают симметрично относительно текста с прописной буквы отдельной строкой. Приложения обозначают заглавными буквами русского алфавита, начиная с А, за исключением букв Ё, З, Й, О, Ч, Ь, Ы, Ъ. После слова —Приложение следует буква, обозначающая его последовательность. Приложение должно иметь заголовок, который пишется симметрично относительно текста с прописной буквы отдельной строкой. Если в отчете одно приложение, то оно обозначается так: Приложение А. Текст каждого приложения, при необходимости, может быть разделен на разделы, подразделы, пункты, подпункты, которые нумеруют в пределах каждого приложения. Перед номером ставится обозначение этого приложения. Рисунки каждого приложения и таблицы обозначаются отдельной нумерацией арабскими цифрами с добавлением перед цифрой обозначения приложения. Например, Рисунок А.5

или Таблица Б.2.

Оформление ссылок

В пояснительной записке должны быть ссылки на все рисунки, таблицы, формулы, приложения, литературные источники, которые приведены в записке. Рисунки, таблицы, формулы располагаются сразу после появления первой ссылки на них, то есть на текущем или следующем листе записки. Порядок номеров приложений и литературных ссылок должен соответствовать порядку появления ссылок на них. При ссылке на литературный источник указывается его порядковый номер, заключенный в квадратные скобки. Например, [4] или [4,5,6]. Ссылка на источник должна ставиться в той части предложения, где начинается пересказываемая мысль другого автора. В обзорах целесообразно использовать ряд устойчивых речевых штампов, например: «...как утверждается в [...]...», «более подробно с...можно познакомиться в []», «В основу этой методики положены идеи, высказанные в [...]» и т.п. Ссылка ставится при первом упоминании материала из источника.

При первой ссылке на рисунок пишется, например, «... в соответствии с рисунком 1.4» или (рисунок 1.4). При первой ссылке на таблицу пишется, например, в таблице 2.3 или (таблица 2.3). Ссылки в тексте на номер формулы дают в скобках, например, ... в формуле (2.5). При ссылке на приложение пишется полностью слово "приложение" и указывается его обозначение, например, "...в приложении А" или (приложение А).

Описание книги четырех и более авторов

Описание книги начинается с заглавия, если она написана четырьмя и более авторами. Авторы пишутся только в сведениях об ответственности. При необходимости их количество сокращают. Также дается описание коллективных монографий, сборников статей. *Примеры:*

Язык программирования С# / А. Хейлмберг, М. Торгерсен, С. Вилтамут, П. Голд – СПб.: Питер, 2012. – 784 с.

Автомобильный транспорт: научная монография; [под ред. Р.М.

Ахмеднабиева]. Новосибирск: Изд. «СибАК», 2013. – 168 с.

Описание статьи из журнала

Для статьи из журнала нужно указать авторов статьи, название статьи, название журнала, год, номер выпуска и страницы начала и окончания статьи.

Пример.

Морозов Е.А. Об устойчивости интегральных кривых в сопряженных пространствах. // Вестник ИжГТУ. – 2005. – №3 – С. 39 – 41.

Ефимов И.Н., Морозов Е.А., Селиванов К.М., Ермолаева Е.В. Каноническое преобразование фазового пространства в динамике твердого тела // Вестник ИжГТУ. – 2009. – №4 – С. 190 – 195.

Учебники, учебные пособия

Для учебников и учебных пособий нужно указать: авторов, название, отметка о типе издания учебное пособие или учебник, город и название издательства, год издания, количество страниц. *Примеры:*

Нетушил А.В. Теория автоматического управления: Нелинейные системы, управление при случайных воздействиях: учебник для вузов. – 2-е изд., перераб. и доп. – М.:Высш. школа, 1983. – 432с.

Богословский С.Г., А.Д. Дорофеев. Динамика полета летательных аппаратов: Учебное пособие. – СПб.: СПбГУАП, 2002. – 64 с.

Описание электронных изданий

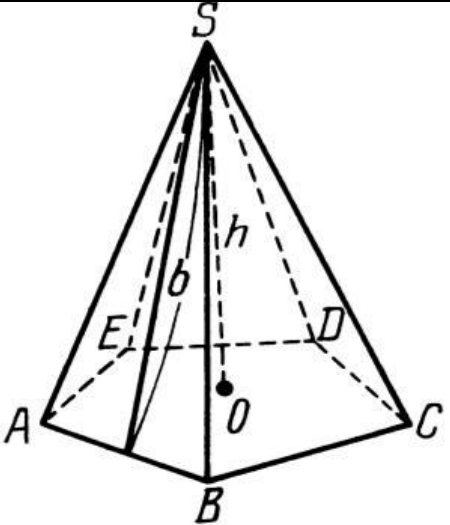
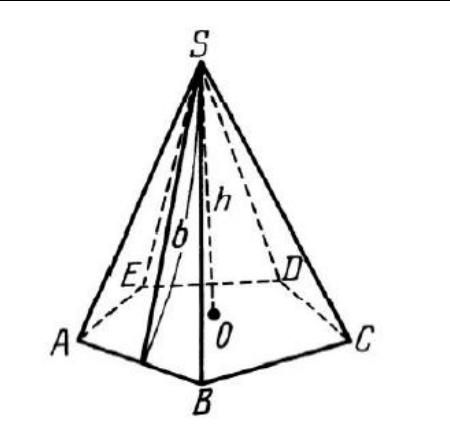
Для электронных источников нужно указать: авторов, название статьи, название сайта (или раздела сайта) и адрес URL. Также должна присутствовать дата обращения к источнику и отметка [Электронный ресурс]. *Примеры:*

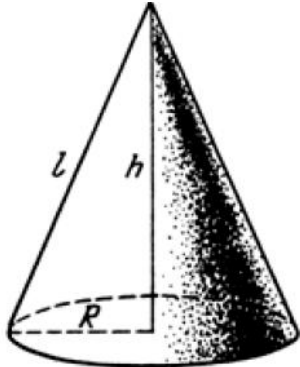
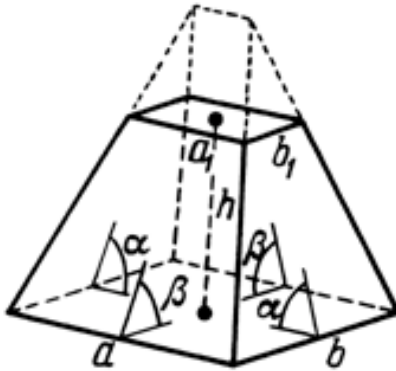
Закирова Э.И. Управление образовательными системами с использованием мультимедийных технологий [Электронный ресурс]. Электронное научно-техническое издание «Наука и образование». URL: <http://technomag.bmstu.ru/doc/606440.html> (дата обращения: 10.09.2021).

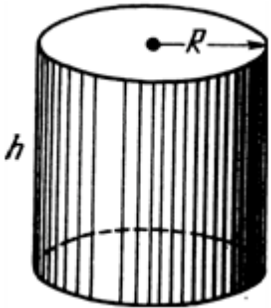
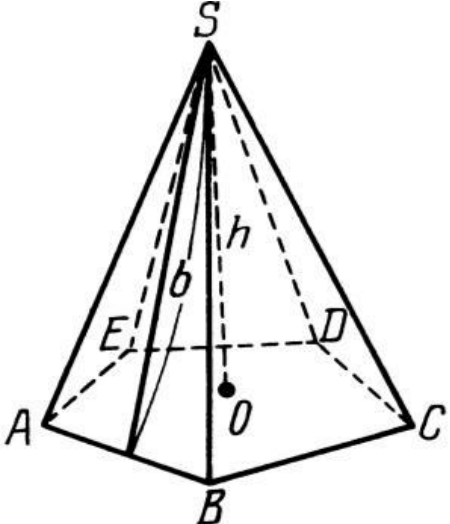
Твердотельный накопитель // Википедия. URL: <http://ru.wikipedia.org/?oldid=69875253> (дата обращения: 10.04.2021).

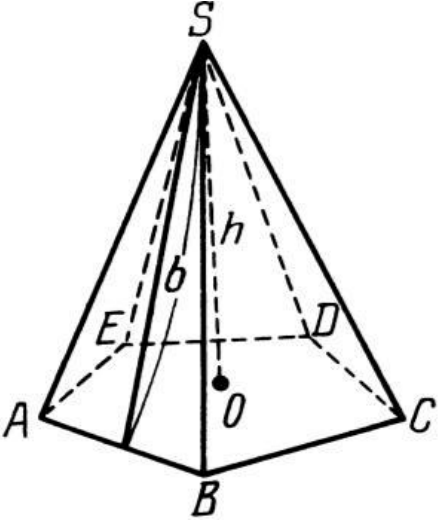
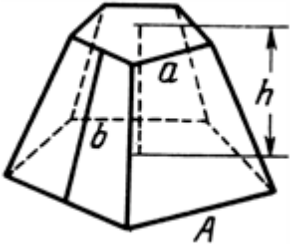
ПРИЛОЖЕНИЕ А

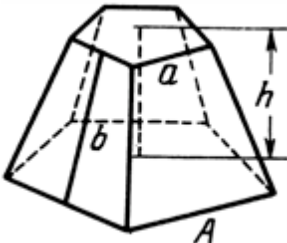
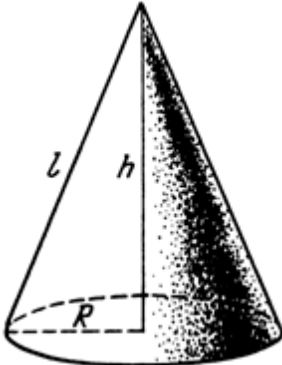
Варианты заданий

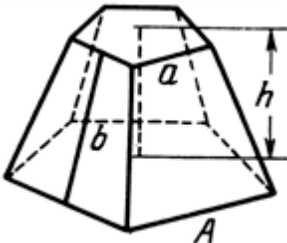
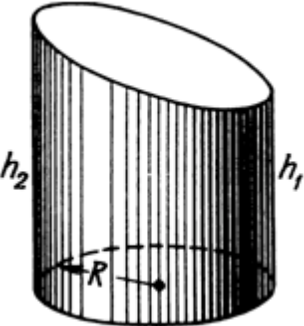
Номер варианта	Теоретическая часть	Практическая (лабораторная) часть	Задание к практической (лабораторной) части	Рисунок
1	Акустические системы	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Пирамида 4 грани	<p>В режиме 640x400 точек на 256 цветов вывести трехмерную фигуру «Пирамида 4 грани». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy.</p>	
2	ЭЛТ - мониторы: техническая характеристика, устройство, принцип работы	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Пирамида 7 граней.	<p>В режиме 800x600 точек на 16 цветов вывести трехмерную фигуру «Пирамида 7 граней». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45</p>	

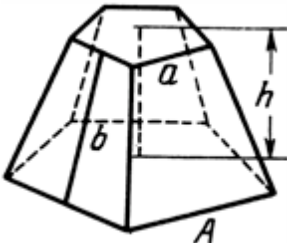
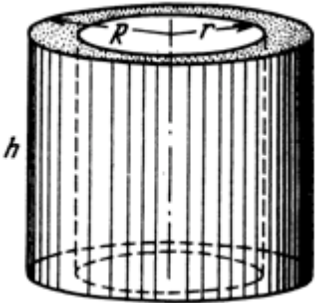
			<p>градусов к оси Z в плоскости перпендикулярной xOy.</p>	
3	<p>Лазерные принтеры: виды, техническая характеристика, устройство, принцип работы.</p>	<p>Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Конус.</p>	<p>В режиме 800x600 точек на 256 цветов вывести трехмерную фигуру «Конус». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy.</p>	
4	<p>Жесткие магнитные диски: техническая характеристика, устройство, принцип работы.</p>	<p>Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Обелиск.</p>	<p>В режиме 1024x768 точек на 16 цветов вывести трехмерную фигуру «Обелиск». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy.</p>	

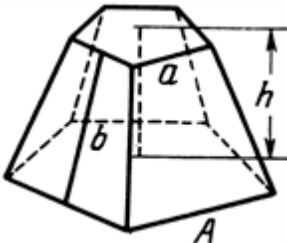
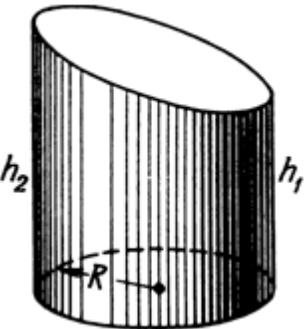
5	Графические адаптеры: принцип действия, основные характеристики.	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Цилиндр.	<p>В режиме 640x480 точек на 32К цветов вывести трехмерную фигуру «Цилиндр». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием predetermined массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy</p>	
6	Многофункциональные устройства	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Пирамида 5 граней.	<p>В режиме 320x200 точек на 16М цветов вывести трехмерную фигуру «Пирамида 5граней». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием predetermined массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy</p>	

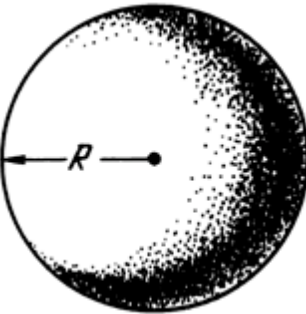
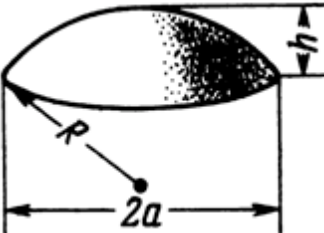
7	Струйные принтеры: виды, техническая характеристика, устройство, принцип работы.	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Пирамида 3 грани.	<p>В режиме 640x480 точек на 16М цветов вывести трехмерную фигуру «Пирамида 3 грани». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy</p>	
8	Твердотельные накопители: техническая характеристика, устройство, принцип работы.	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Усеченная пирамида 3 грани.	<p>В режиме 1280x1024 точек на 16 цветов вывести трехмерную фигуру «Усеченная пирамида 3 грани». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy.</p>	

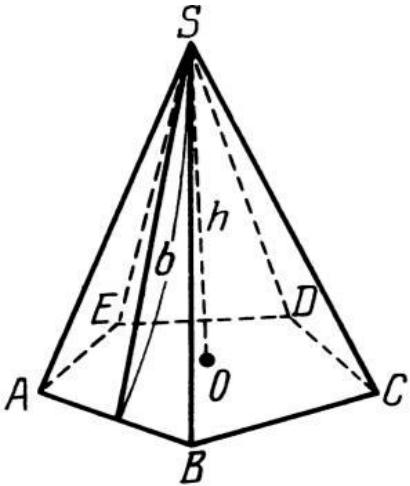
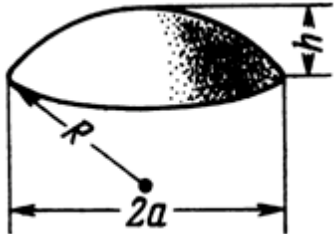
9	Плазменные мониторы: техническая характеристика, устройство, принцип действия.	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Усеченная пирамида 3 грани.	В режиме 1280x1024 точек на 256 цветов вывести трехмерную фигуру «Усеченная пирамида 9 граней». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy.	
10	Звуковые карты: принцип действия, основные характеристики.	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Усеченный конус	В режиме 1024x768 точек на 256 цветов вывести трехмерную фигуру «Усеченный конус». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy.	

11	Flash-память. Принцип действия, основные характеристики, область применения.	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Усеченная пирамида 7 граней.	В режиме 640x480 точек на 64К цветов вывести трехмерную фигуру «Усеченная пирамида 7 граней». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy.	
12	Системы охлаждения: необходимость использования, виды.	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Усеченный цилиндр	В режиме 800x600 точек на 64К цветов вывести трехмерную фигуру «Усеченный цилиндр». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy.	

13	Дисковые массивы RAID: основные характеристики, область применения, стандарты.	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Усеченная пирамида 5 граней.	<p>режиме 800x600 точек на 256 цветов вывести трехмерную фигуру «Усеченная пирамида 5 граней». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy.</p>	 <p>The diagram shows a truncated pentagonal pyramid. The top pentagonal face is labeled 'a', the bottom pentagonal face is labeled 'b', and the vertical height is labeled 'h'. The front face is labeled 'A'.</p>
14	Светодиодные принтеры: виды, техническая характеристика, устройство, принцип работы.	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Полный цилиндр	<p>В режиме 1280x1024 точек на 256 цветов вывести трехмерную фигуру «Полый цилиндр». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy</p>	 <p>The diagram shows a hollow cylinder. The outer radius is labeled 'R', the inner radius is labeled 'r', and the height is labeled 'h'.</p>

15	Акустические системы: виды, техническая характеристика, устройство	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Усеченная пирамида 4 грани	<p>В режиме 1024x768 точек на 256 цветов вывести трехмерную фигуру «Усеченная пирамида 4 грани». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy.</p>	
16	Системные платы: виды, техническая характеристика, устройство	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Усеченный цилиндр	<p>В режиме 1024x768 точек на 16 цветов вывести трехмерную фигуру «Усеченный цилиндр». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy</p>	

17	Твдротелые накопители: техническая характеристика, устройство, принцип работы	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Сфера	<p>В режиме 640x480 точек на 256 цветов вывести трехмерную фигуру «Сфера». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy.</p>	 <p>A diagram of a sphere with a radius R indicated by a horizontal line from the center to the left edge.</p>
18	Жидко - кристаллические мониторы: техническая характеристика, устройство, принцип работы.	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Шаровой сегмент	<p>В режиме 800x600 точек на 256 цветов вывести трехмерную фигуру «Шаровой сегмент». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy.</p>	 <p>A diagram of a spherical cap. The radius of the sphere is labeled R. The height of the cap is labeled h. The diameter of the base is labeled $2a$.</p>

19	Лазерные принтеры: виды, техническая характеристика, устройство, принцип работы.	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Пирамида 9 граней	<p>В режиме 1280x1024 точек на 16 цветов вывести трехмерную фигуру «Пирамида 9 граней». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается вдоль оси X. Вращение осуществлять под углом 45 градусов к оси Z в плоскости перпендикулярной xOy.</p>	 <p>The diagram shows a 3D representation of a nonagonal pyramid. The apex is labeled 'S'. The base is a nonagon with vertices labeled 'A', 'B', 'C', 'D', and 'E'. A vertical dashed line from 'S' to the center of the base 'O' represents the height 'h'. A solid line from 'S' to vertex 'B' is labeled 'b'. The base is shown in perspective, with some edges as dashed lines to indicate they are hidden.</p>
20	Сканеры: виды, техническая характеристика, устройство, принцип работы.	Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура – Шаровой сегмент	<p>В режиме 640x400 точек на 256 цветов вывести трехмерную фигуру «Шаровой сектор». Написать программу вращения фигуры в плоскости с шагом 15 градусов против часовой стрелки. Вывод точек осуществлять непосредственно в видеопамять с использованием предопределенного массива и встроенного Ассемблера. Видеорежим устанавливать, используя библиотеку DOS Turbo Pascal и встроенный Ассемблер. Фигура располагается</p>	 <p>The diagram shows a 3D representation of a spherical cap. The cap is shaded to show its curved surface. The radius of the sphere is labeled 'R'. The height of the cap is labeled 'h'. The diameter of the base of the cap is labeled '2a'. The cap is shown sitting on a rectangular base.</p>

ПРИЛОЖЕНИЕ Б

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Чайковский филиал

федерального государственного автономного
образовательного учреждения высшего образования

«Пермский национальный исследовательский политехнический университет»
(ЧФ ПНИПУ)

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

по дисциплине ЭВМ и периферийные устройства

Вариант №__

Ф.И.О. студента _____

Специальность, индекс группы _____

Наименование темы:

Теоретическая часть: _____

Практическая (лабораторная) часть: Работа с прерываниями в библиотеке DOS и Ассемблере. Фигура - _____

1 Цель курсовой работы

Целью курсовой работы является исследование структуры и характеристик ЭВМ и периферийных устройств, приобретение практических навыков в определении неисправностей периферийных устройств и путей устранения и предотвращения их возникновения, а также ознакомиться с функционированием ЭВМ на уровне взаимодействия аппаратных компонентов и получить навыки управления ими, написав и отладив программу на языке низкого уровня (Ассемблер).

2 Задачи курсовой работы

Основные задачи:

- 2.1 описание предмета исследования;
- 2.2 построение классификации и описание принципа функционирования;
- 2.3 описание конструктивных особенностей и характеристик различных видов;
- 2.4 построение классификации неисправностей предмета исследования;
- 2.5 определение методы диагностики, устранения, а также профилактики возникновения перечисленных неисправностей;
- 2.6 ознакомление с программированием видеоадаптеров в нестандартных режимах;
- 2.7 ознакомление с организацией прерываний в ЭВМ;
- 2.8 построение математической модели трехмерной фигуры;
- 2.9 проведение алгоритмизации и написание программы установки нестандартных режимов видеоадаптеров.

3 Структура пояснительной записки

Курсовая работа состоит из трех частей:

- 3.1 Теоретические сведения о предмете исследования:

Первая часть курсовой работы заключается в поиске, анализе и обработке информации по указанной теме и представляет собой реферативное сообщение о предмете исследования. Должна содержать сведения, актуальные на сегодняшний день и

описывающие предмет исследования с позиций классификации, конструктивных особенностей, принципов функционирования и применения в составе вычислительной техники.

3.2 Практические сведения о неисправностях предмета исследования и методов их устранения:

Вторая часть курсовой работы является практической частью и заключается в структурировании сведений о неполадках, которые могут возникнуть в процессе функционирования предмета исследования и методах их устранения.

3.3 Работа с прерываниями:

Третья часть курсовой работы заключается в ознакомлении с функционированием ЭВМ на уровне взаимодействия аппаратных компонентов, и получить навыки управления ими. Ознакомиться с программированием видеоадаптеров в нестандартных режимах, ознакомиться с организацией прерываний в ЭВМ. Написать и отладить программу на языке программирования с применением встроенного ассемблера.

4 Задание на лабораторную часть курсовой работы

5 Состав документации

5.1 Титульный лист.

5.2 Содержание.

5.3 Введение

5.4 Теоретические сведения о предмете исследования.

5.5 Сведения о неисправностях предмета исследования и их устранении.

5.6 Работа с прерываниями.

5.7 Заключение.

5.8 Список литературы.

5.9 Приложения.

Срок представления работы (проекта) к защите « ____ » _____ 202_ г.

Руководитель работы (проекта) _____
(подпись, дата) (фамилия, имя, отчество)

Задание принял к исполнению _____
(подпись, дата) (фамилия, имя, отчество)

ПРИЛОЖЕНИЕ Б

Министерство науки и высшего образования Российской Федерации
Чайковский филиал
Федерального государственного автономного образовательного учреждения
высшего образования

Пермский национальный исследовательский
политехнический университет

Кафедра автоматизации, информационных и инженерных технологий

Курсовая работа *«название работы»*

по дисциплине
«ЭВМ и периферийные устройства»

Выполнил студент группы _____

Проверил ст. преподаватель:

Сухих И.И.