Министерство образования и науки
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ГОУВПО «АмГУ»)

УТВЕРЖДАЮ
Зав.кафедрой ИиУС
_____А.В.Бушманов
«____»_____2009г.

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ

Современные программно-технические средства автоматизированных систем научных исследований
для магистров направления
23010068 – «Информатика и вычислительная техника» по магистерской программе «Компьютерное моделирование»

Составитель: А.В.Бушманов

2009

Печатается по решению
редакционно-издательского совета
факультета математики
и информатики
Амурского государственного
университета

Современные программно-технические средства автоматизированных систем научных исследований для магистров направления 23010068 – «Информатика и вычислительная техника» магистерской программы «Компьютерное моделирование»: учебно-методический комплекс дисциплины. /Бушманов А.В. – Благовещенск. Изд-во Амурского гос. ун-та, 2009г.

ОГЛАВЛЕНИЕ

1. ВЫПИСКА ИЗ ГОСУДАРСТВЕННОГО ОБРАЗОВАТЕЛЬНОГО СТАНДАРТА ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

Направление подготовки магистранта
23010068 – Информатика и вычислительная техника

Магистерская программа

Компьютерное моделирование

Квалификация – *магистр*.

Целью курса является изучение современных микропроцессорных и компьютерных систем. Данные системы рассматриваются с позиций системного анализа и представляются в виде многоуровневых иерархических систем. Поэтому при изучении курса основное внимание уделяется стратифицированному представлению микропроцессорных и компьютерных систем, проектированию их архитектуры, организации, обеспечению заданных критериев качества и т.п.

В результате изучения этого курса магистранты должны иметь представления об особенностях проектирования компьютерных и микропроцессорных систем, уметь проектировать математическое, алгоритмическое, программное и техническое обеспечение при создании сложных систем контроля, управления, связи и т.п., требующих системного подхода.

Дисциплина «Современные программно-технические средства автоматизированных систем научных исследований» базируется на курсах «Информатика», «Программирование», «Теория автоматического управления», «Измерительная техника и датчики», «Электронные цепи и

микросхемотехника», «Основы микропроцессорной техники» и является завершающей в подготовке магистров.

РАБОЧАЯ ПРОГРАММА

по дисциплине: Современные программно-технические средства автоматизированных систем научных исследований

Для направления 23010068 – «Информатика и вычислительная техника» магистерской программы «Компьютерное моделирование»

Факультет: Математики и информатики
Кафедра: Информационных и управляющих систем

Курс шестой

Семестр: 11

Учебный план набора 2009 года и последующих лет

Распределение учебного времени

Лекции	18 час.
Лабораторные занятия	36 час.
Всего ауд. занятий	51 час.
Самостоятельная работа	86 час.
Контроль сам.работы	10
Общая трудоемкость	150 час.

Зачет 9 семестр

1. Цель и задачи дисциплины, ее место в учебном процессе

Целью данного курса является обучение магистрантов построению автоматизированных систем для научных исследований.

В результате изучения дисциплины магистранты должны:

- знать основные структуры и виды обеспечений автоматизированных систем, типы и характеристики первичных измерительных преобразователей, структуру ЭВМ и организацию обмена данными с внешними устройствами, архитектуру и алгоритмы функционирования универсальных измерительных интерфейсов;
- уметь разработать автоматизированную систему на базе стандартного приборного интерфейса, выбрать необходимые технические и программные средства, разработать алгоритмы измерения и обработки данных, отладить программное обеспечение создаваемой системы.

2. Содержание дисциплины

2.1. Федеральный компонент
ДД Цикл дополнительных дисциплин

2.2. Лекционные занятия

2.2.1. Введение. Структура, содержание и задачи курса "Современные программно-технические средства автоматизированных систем научных исследований", его связь с другими дисциплинами. (2 часа).

2.1.2. Тема 1. Структуры и виды обеспечений автоматизированных систем для научных исследований (2 часа).

2.1.3. Тема 2. Датчики автоматизированных систем (2 часа).

2.1.4. Тема 3. Структура ЭВМ (4 часа).

2.1.5. Тема 4. Обмен данными с внешними устройствами ЭВМ (2 часа).

2.1.6. Тема 5. Системы сопряжения ЭВМ с объектом автоматизации (2 часа).

2.1.7. Тема 6. Интерфейсная шина с байт-последовательным, бит-параллельным обменом информацией (2 часа).

2.1.8. Тема 7. Основные тенденции и направления развития автоматизированных систем (2 часа).

2.3. Лабораторные занятия

2.3.1. Лабораторная работа №1. Аналого-цифровые и цифроаналоговые преобразователи (4 часа).

2.3.2. Лабораторная работа №2. Программное обеспечение АСНИ (4 часа).

2.3.3. Лабораторная работа №3. Схемы включения измерительных преобразователей (4 часа).

2.3.4. Лабораторная работа №4. Программно-управляемый режим внешних устройств (6 часов).

2.3.5. Лабораторная работа №5. Организация обмена данными на системной шине (4 часа).

2.3.6. Лабораторная работа №6. Особенности организации промышленных компьютеров (6 час.)

2.3.7. Лабораторная работа № 7. Программно-доступные регистры контроллера и их организация (4 часа).

2.3.8. Лабораторная работа № 8. Адресация регистров контроллера и их модулей (4 часа).

2.4. Самостоятельная работа

Распределение времени по видам самостоятельных работ приведено в таблице 1.

Таблица 1 – Виды самостоятельных работ

№	Наименование работ	Кол-во часов	Форма отчетности
1	Проработка лекционного материала	10	зачет
2	Подготовка к лабораторным занятиям	10	зачет
3	Подготовка реферата	30	реферат, защита
4	Выполнение индивидуальных заданий	40	отчет, защита

2.5. Примерные темы рефератов

2.5.1 Квантование непрерывного сигнала.

2.5.2 Примеры автоматизированных систем для научных исследований.

2.5.3 Технические средства автоматизации эксперимента.

2.5.4 Программное обеспечение АС.

2.5.5 Классификация датчиков и основные требования к ним.

2.5.6 Схемы включения, функции, преобразования и погрешности.

2.5.7 Датчики сил и механических напряжений.

2.5.8 Фоточувствительные датчики.

2.5.9 Датчики давления.

2.5.10 Система памяти. Иерархия, особенности подсистем.

2.6. Примерные темы индивидуальных заданий

2.6.1 Архитектурные особенности организации ЭВМ различных классов.

2.6.2 Основные направления в архитектуре процессоров.

2.6.3 Макросредства языка ассемблера.

2.6.4 Создание Windows-приложений на ассемблере.

2.6.5 Работа с окнами диалога Windows на ассемблере.

2.6.6 Использование ассемблера для разработки программ, управляющих аппаратными ресурсами ПК.

2.6.7 Связь ассемблерных программ с программами, разработанными с использованием языков высокого уровня.

2.8. Методика формирования текущего рейтинга

Лабораторные занятия, предусмотренные рабочей программой, выполняются согласно расписанию занятий. Собеседование проводится во время экзаменационной сессии. Рейтинг выстроен так, что для получения оценки студенту предоставляется возможность добрать баллы, после выполнения и защиты индивидуальных заданий, лабораторных работ и рефератов на собеседовании. Собеседование предполагает устную проверку знаний студента.

Максимальный рейтинг по дисциплине составляет 120 баллов и определяется по таблице 2. Для получения оценки «отлично» требуется набрать не менее 100 баллов, «хорошо» – 80 баллов.

Таблица 2 – Распределение максимального рейтинга по элементам контроля

№	Виды контроля	Начало (неделя)	Окончание (неделя)	Максим. балл
1	Посещение лекций	1	17	30
2	Лабораторная работа №1	1	2	5
3	Лабораторная работа №2	2	3	5
4	Лабораторная работа №3	3	4	5
5	Лабораторная работа №4	4	5	5
6	Лабораторная работа №5	5	6	5
7	Лабораторная работа №6	6	7	5
8	Лабораторная работа №7	7	8	5
9	Лабораторная работа №8	8	9	5
10	Реферат	1	8	20
11	Индивидуальное задание	5	16	20
12	Собеседование	17	17	10
13	Всего баллов	1	17	120

Выполнение заданий в неустановленный срок приводит к снижению рейтинга.

3. Литература

3.1. Основная литература:

3.1.1 Угрюмов Е.П. Цифровая схемотехника: учеб.пособие: рек. УМО/ Е.П.Угрюмов. -2-е изд., перераб. и доп. –СПб.: БХВ-Петербург, 2007. -2007. -782 с.

3.1.2 Грушвицкий Р.И. Проектированиесистем на микросхемах с программируемой структурой: (учеб.пособие)/ Р.И.Грушвицкий, А.Х.Мусаев, Е.П.Угрюмов. -2-е изд. –СПб.: БХВ-Петербург, 200. -736 с.

3.1.3 Фрайден Дж. Современные датчики: справ./ Дж.Фрайден; пер. с англ. Ю.А.Заболотной, под ред. Е.Л.Свинцова. –М.: Техносфера, 2006. -589 с.

3.1.4 Курицын С.А. Телекомуникационные технологии и системы: учеб. пособие/ С.А.Курицын. –М.: Академия, 2008. -300 с.

3.1.5 Рудометов Е.А. Материнские платы и чипсеты/ Е.А.Рудометов. -4-е изд. –СПб:Питер, 2007. -368 с.

3.1.6 Схемотехника электронных систем. Аналоговые и импульсные устройства: (учеб. пособие)/ В.И.Бойко и др. –СПб.: БХВ-Петербург, 2004. -482 с.

3.1.7 Схемотехника электронных систем. Цифровые устройства: (учеб. пособие)/ В.И.Бойко и др. –СПб.: БХВ-Петербург, 2004. -497 с.

3.2. Дополнительная литература:

3.2.1 Юров В.И. Assembler: учеб.: доп. Мин.обр. РФ/ В.И.Юров. -2-е изд. –СПб.:Питер, 2008. -637 с.

3.2.2. Управляющие системы и автоматика: производственно-практическое изд./ Д.Шмид и др.: пер.с нем.Л.Н.Казанцева. –М.:Техносфера, 2007. -584 с.

3.2.3 Шандров Б.В. Технические средства автоматизации: учеб.: рек. Мин.обр. РФ/ Б.В.Шандров, А.Д.Чудаков. –М.:Академия, 2007 _362 с.

3.2.4 Свердлов С.З. Языки программирования и методы трансляции: учеб.пособие: доп. Мин.обр. РФ/ С.З.Свердлов. –СПб.:Питер, 2007. -638 с.

3.2.5 Бройдо В.Л. Архитектура ЭВМ и систем: учеб: рек. Мин.обр. и науки РФ/ В.Л.Бройдо, О.П.Ильина. -2-е изд. – СПб.:Питер, 2009. -720 с.

3.2.6 Марголин В.И. Физические основы микроэлектроники: учеб.: рек. УМО/ В.И.Марголин, В.А.Жабрев, В.А.Тупик. –М.: Академия, 2008. -400 с.

3.2.7 Наноэлектроника/ под ред. А.А.Орликовского. –М.: Изд-во Моск. Гос.техн. ун-та им.Н.Э.Бауман. -2009. -720 с.

4. Учебно – методическая (технологическая) карта дисциплины

Номер недели	Номер темы	изучаемые на лекцииВопросы,	Занятия		и методические пособияИспользуемые наглядные	Самостоятельная работа магистрантов		Форма контроля
			Практические	Лабораторные		Содержание	Часы	
1	2	3	4	5	6	7	8	9
1	Вв[1]	2.2.1	-	2.3.1	3.1.1	3.2.1	10	злр [2]
2	2	2.2.2	-	2.3.1	3.1.2	3.2.2	10	
3	3	2.2.3	-	2.3.2	3.1.3	3.2.3	10	злр
4	4	2.2.4	-	2.3.2	3.1.4	3.2.4	10	
5	4	2.2.4	-	2.3.3	3.1.3	3.2.3	10	
6	5	2.2.5	-	2.3.3	3.1.5	3.2.4	10	злр
7	6	2.2.6	-	2.3.4	3.1.1	3.2.2	10	
9	7	2.2.7	-	2.3.4	3.1.4	3.2.3	10	
8	8	2.2.8	-	2.3.4	3.1.5	2.2.2	9	зач.[4]

1. Введение,
2. Защита отчета о выполнении лабораторной работы,
3. Зачет по лабораторным работам,
4. Собеседование по результатам.

3. ГРАФИК САМОСТОЯТЕЛЬНОЙ РАБОТЫ МАГИСТРАНТОВ

4.

№	Наименование работ	Кол-во часов	Форма отчетности
1	Проработка лекционного материала	10	зачет
2	Подготовка к лабораторным занятиям	10	зачет
3	Подготовка к практическим занятиям	30	реферат, защита
4	Выполнение индивидуальных заданий	40	отчет, защита

4. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ САМОСТОЯТЕЛЬНОЙ РАБОТЫ МАГИСТРАНТОВ

В течение семестра магистрантам даются для самостоятельного рассмотрения и изучения темы, необходимые для выполнения лабораторных работ.

По каждой лабораторной работе магистранты выполняют отчет, требования к выполнению отчета указаны в методических разработках к лабораторным работам.

- В течение семестра магистранты выполняют контрольные работы по тематике предложенной в рабочей программе. Подготовка к контрольной работе предусматривает изучение материалов лекции и демонстрацию умения решать предложенные задачи в контрольной работе.
-

5. ПЕРЕЧЕНЬ УЧЕБНИКОВ И УЧЕБНЫХ ПОСОБИЙ

5.1. Клейменов М.С. Системный подход к проектированию сложных систем //Журнал д-ра Добба. 1999. – №1. – С.9-14.

5.2. Норенков И.П. Основы автоматизированного проектирования, М. Изд. МГТУ им. Баумана, 2000г.

5.3. Серия ГОСТ 34.ХХХ "Информационная технология. Комплекс стандартов на автоматизированные системы".

5.4. ГОСТ Р ИСО/МЭК 12207-99 "Информационная технология. Процессы жизненного цикла программных средств".

5.5. РД 50–680–88 Методические указания. Автоматизированные системы. Основные положения.

5.6. Э.Е. Кудряшова. САПР ТП как система интегрированной среды. Интегриро-ванные автоматизированные системы CAD/CAM/CAE: Учеб. пособие / Волгоград: ВолгГТУ, 1998. - 148с.

5.7. Э.Е. Кудряшова. Новые информационные технологии в автоматизированных системах обработки ннформации и управления: Учеб. пособие/ Волгоград: ВолгГТУ, 2000г. -88с.

5.8. Маклаков С.В. BPWin, ERWin. CASE – средства разработки информационных систем. –М.: ДИАЛОГ-МИФИ, 1999.

5.9. Грекул В.И., Денищенко Г.Н., Коровкина Н.Л. Проектирование информационных систем. /Интернет-университет информационных технологий - ИНТУИТ.ру, 2005.

5.10. Амриш К.И., Ахмед Х.З. Разработка корпоративных Java-приложений с использованием J2EE и UML. /Пер. с англ. -М.: "Вильямс", 2002. - 272 с.

5.11. Боггс У., Боггс М. UML и Rational Rose./ - М.: "ЛОРИ", 2000. - 582 с.

5.12. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++. - М.: "Бином", СПб: "Невский диалект", 1999. - 560 с.

5.13. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя. /- М.: ДМК, 2000. - 432 с.

5.14. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. /Паттерны проектирования. - СПб: "Питер", 2001.- 368 с.

6. КРАТКИЙ КОНСПЕКТ ЛЕКЦИЙ

Электронная вычислительная машина - комплекс технических и программных средств, предназначенный для автоматизации подготовки и решения задач пользователей. Под пользователем понимают человека, в интересах которого проводится обработка данных на ЭВМ. В качестве пользователя могут выступать заказчики вычислительных работ, пользователей к выполнению вычислительных работ удовлетворяются специальным подбором и настройкой технических и программных средств. Обычно эти средства взаимосвязаны и объединяются в одну структуру.

Структура - совокупность элементов и их связей. Различают структуры технических, программных и аппаратурно-программных средств. Выбирая ЭВМ для решения своих задач, пользователь интересуется функциональными возможностями технических и программных модулей (как быстро может быть решена задача, насколько ЭВМ подходит для решения данного круга задач, какой сервис программ имеется в ЭВМ, возможности диалогового режима, стоимость подготовки и решения задач и т.д.). При этом пользователь интересуется не конкретной технической и

программной реализацией отдельных модулей, а более общими вопросами возможности организации вычислений. Последнее включается в понятие архитектуры ЭВМ, содержание которого достаточно обширно.

Архитектура ЭВМ - это многоуровневая иерархия аппаратурно-программных средств, из которых строится ЭВМ. Каждый из уровней допускает многовариантное построение и применение. Конкретная реализация уровней определяет особенности структурного построения ЭВМ. В последующих разделах учебника эти вопросы подробно рассматриваются.

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

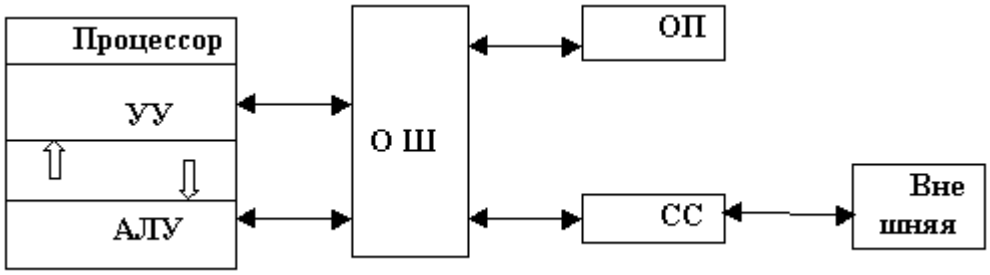
Самого же пользователя интересуют обычно более общие вопросы, касающиеся его взаимодействия с ЭВМ (человеко-машинного интерфейса), начиная со следующих групп характеристик ЭВМ, определяющих ее структуру:

технические и эксплуатационные характеристики ЭВМ (быстродействие и производительность, показатели надежности, достоверности, точности, емкость оперативной и внешней памяти, габаритные размеры, стоимость технических и программных средств, особенности эксплуатации и др.);

характеристики и состав функциональных модулей базовой конфигурации ЭВМ; возможность расширения состава технических и программных средств; возможность изменения структуры;

состав программного обеспечения ЭВМ и сервисных услуг (операционная система или среда, пакеты прикладных программ, средства автоматизации программирования).

Специальные ЭВМ (сЭВМ) ориентированы на решение специальных вычислительных задач либо задач управления, решаемых в режиме реального времени Последние используются в различных системах управления и часто образуют специальные ГВМ, обрабатывающие аналого-цифровую информацию.Общая архитектура сЭВМ приведена на рис.



Общая архитектура специальной ЭВМ (сЭВМ)

Основные компоненты сЭВМ:

- процессор, содержащий устройство управления (УУ) и арифметико-логической устройство (АЛУ), сложность и функциональные возможности которых определяются назначением данного типа машины;

- оперативной памяти (ОП);

- система сопряжения (СС), предназначенная для обеспечения интерфейса между памятью процессора и внешней средой, в качестве которой выступают различные источники и приемники информации (датчики, управляющие блоки, оконечные устройства и т.д.). Так как внешняя среда оперирует как с дискретной, так и с аналоговой информацией, то и СС в общем случае обеспечивает интерфейс между обеими формами информации различного типа (данные, логическая, управляющая и др.).

Основные компоненты сЭВМ объединяются общей шиной (ОШ), или общей магистралью.

Процессор функционирует под управлением программы, находящейся в ОП или в постоянном запоминающем устройстве (ПЗУ), и его архитектура определяется, во многом, классом машины и спецификой решаемых задач. Если сЭВМ ориентирована на решение узкого класса задач по единой программе, но с различными данными, то программу можно поместить в ПЗУ, хранящее информацию даже при отключенном питании; ее в ряде случаев можно реализовать и аппаратно, т.е. "зашить" в электронные схемы машины. В последнем случае повышаются надежность и быстродействие сЭВМ. Такой подход широко используется, например, во многих видах бортовой ВТ, работающей в жестком режиме реального времени. В зависимости от класса сЭВМ обеспечиваются языками программирования различного уровня (от микропрограммного до высокого).

Повышенные требования к производительности сЭВМ (в ряде случаев не менее 100 млн оп/с) и спецификация решаемых задач стимулировали использование неклассической не - неймановской архитектуры организации вычислительного процесса.

Основными направлениями развития класса сЭВМ являются работы по созданию оптических спецпроцессоров для обработки изобразительной информации и сигналов. Перспективным направлением разработки специальной ВТ нетрадиционной архитектуры можно считать мультипроцессорные, распределенные и иерархические системы и сети, образующие уже комплексы и системы сЭВМ.

Микропроцессорные ЭВМ и ПК.

Создание в начале 70-х годов первых универсальных МП можно считать началом эры микро-ЭВМ, а затем и персональных компьютеров (ПК). Первый из этой серии МП Intel-4004 выполнял все функции центрального процессора ЭВМ общего назначения и в совокупности с четырьмя микросхемами (памяти, УУ и интерфейса ввода/вывода) представлял собой компьютер, не уступающий по мощности большим ЭВМ середины 50-х г. 20 века.

Все ЭВМ можно условно разделить на четыре типа:

1. микро (micro)-;
2. мини(mini)-;
3. общего назначения (mainframe)-;
4. супер (super)-ЭВМ.

В настоящее время под микро-ЭВМ понимается микропроцессорная ВТ, используемая двояко:

1. в качестве универсального блока обработки данных и/или управления, выпускаемого в виде одной БИС/СБИС и предназначенного для встраивания в различные специализированные системы контроля и управления, и в другую технику для расширения ее функциональных и интеллектуальных возможностей, производительности, надежности, а также улучшения существующих и придания ей новых свойств;

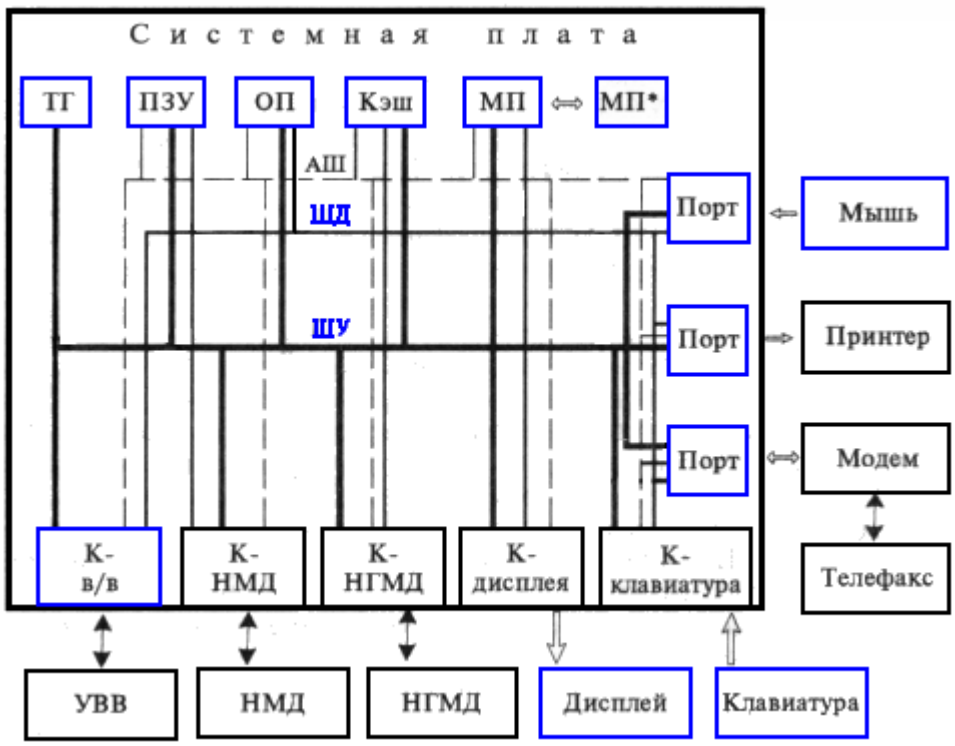
2. в качестве надежной малогабаритной ЭВМ персонального пользования (ПК), предназначен для работы в интерактивном режиме, имеющей развитое ПО различного назначения и ориентированной на самый широкий круг пользователей различных возрастов и профессий, доступной по ценам массовому пользователю;

Практически все современные универсальные (относительно команд) микро-ЭВМ отражают классическую неймановскую архитектуру.

Большинство современных ЭВМ в своем составе имеют в качестве внешней памяти (ВП) накопители на различного типа магнитных носителях (НМД, НГМД, НМЛ и др.); Для вывода и документирования данных и программ используются различного типа печатающие устройства (принтеры) и рисующие плоттеры, а для работы в системах телекоммуникации ЭВМ на основе модемов имеют возможность обмена информацией с удаленными телефонными абонентами (например, с телефаксами или информационно-вычислительных сетях).

Системная плата ПК (рис.) содержит следующие основные компоненты:

- тактовый генератор (ТГ),
- постоянное запоминающее устройство (ПЗУ),
- оперативную память (ОЗУ),
- микропроцессор (МП) {и возможно, сопроцессор (МП)},
- контроллеры передачи данных,
-контроллеры ввода/вывода и порты ввода/вывода, а также шины управления, адресации и данных, образующие в совокупности общую шину



Структурная схема системной платы ПК.

Главной частью системной платы является МП, управляющий работой всей системы узлов ПК и программой, описывающей алгоритм решаемой задачи. МП имеет сложную структуру, реализованную в виде системы электронных логических схем; в качестве основных его компонент можно выделить:

- арифметико-логическое устройство (АЛУ);
- устройство управления (УУ);
- систему прерываний (СПр);
- устройство управления общей шиной (УОШ) - системным интерфейсом;

специальные регистры.

Для расширения возможностей и повышения функциональных характеристик МП дополнительно может поставляться специальный сопроцессор (МП), как правило, служащий для расширения набора команд ведущего МП. В последнее время в качестве сопроцессоров при создании мультипроцессорных систем с однотипными МП и аппаратной поддержкой вычислительных процессоров все чаще используются транспьютеры, представляющие собой МП специального типа. Особенностью транспьютеров является наличие быстрых коммуникационных каналов связи, каждый из которых может одновременно передавать по одной магистрали данные в МП, а по другой - из него. В составе команд транспьютеров имеются команды управления процессами, поддержки предложений языков программирования высокого уровня (Fortran, Pascal, C). Высокая производительность транспьютеров обусловлена высокими скоростями передачи операндов команд в АЛУ и их обработки в нем. Типичными примерами транспьютеров являются известные модели T414 и T800 фирмы

INMOS. В настоящее время наиболее распространенными и используемыми для создания ПК различных моделей являются 32-битные МП фирм Intel, Motorola, DEC, AND (США), NEC (Япония) и INMOS (Англия).

Кэш-память (Кэш) с малым временем доступа служит для временного хранения промежуточных результатов и содержимого наиболее часто используемых ячеек ОП и регистров МП; объем Кэш-памяти зависит от модели ПК и для большинства моделей составляет 256 Кбайт.

Контроллер ввода/вывода является необязательным и обычно применяется в многопользовательских системах; он берет на себя управление некоторыми операциями по вводу/выводу, при его отсутствии выполняемыми самим МП. Контроллеры внешних устройств служат для обеспечения прямой связи вторых с ОП, минуя МП (режим прямого доступа к ОП); как правило, они используются для устройств быстрого обмена данными с ОП (НГМД, НМД, дисплей и др.), а также для обеспечения работы в групповом и сетевом режимах.

Порты ввода/вывода служат для обеспечения обмена информацией ПК с внешними, не очень быстрыми устройствами (клавиатура, мышь, джойстик, телефонная сеть и др.). Порты бывают входными, выходными и универсальными (ввод/вывод), а также последовательными и параллельными. Последовательный порт ведет побитный, а параллельный - побайтный обмен информацией; поэтому принтер подключается к параллельному порту, а телефонная линия связи (через модем) - к последовательному. Большинство современных ПК имеет один параллельный и два последовательных порта ввода/вывода. Информация, поступающая через порт, направляется сначала в МП, а затем в ОП и наоборот. Так как клавиатура, мышь и дисплей имеют свои выделенные участки в ОП, то эти устройства связаны с системной платой контроллерами, хотя клавиатура и мышь являются достаточно медленными устройствами ввода (однако они обеспечивают непосредственный интерфейс с ПК).

Наконец, все узлы системной платы (рис.) связаны системным интерфейсом (СИ) типа "общая шина", организация которого зависит от модели и типа ПК и микро-ЭВМ. Он представляет собой систему линий передачи адресов, данных и управляющих сигналов различного типа и назначения.

Общие сведения о мини-ЭВМ. Общая архитектура и состав мини-ЭВМ.

Составляют достаточно малый класс, занимающий промежуточное положение между классами микро-ЭВМ (ПК) и ЭВМ общего назначения; мини-ЭВМ имеют ОП объемом порядка от 100 Мбайт до нескольких Гигабайт, приближаясь по вычислительным возможностям к ЭВМ общего назначения. В этом отношении среди мини-ЭВМ даже выделяют подкласс супер- мини-ЭВМ. В связи с развитием элементной базы, во многом общей для ЦВТ, традиционных и перспективных архитектур грань между классами ЭВМ становится весьма размытой и во многом начинает носить условный характер.

Характерной чертой современных мини-ЭВМ можно считать развитый многопользовательский режим доступа к вычислительным ресурсам, соответствующим по возможностям ЭВМ общего назначения среднего класса.

Современное архитектурное развитие мини-ЭВМ строится на использовании идей мультипроцессорности, параллельной обработки и RISC- подхода (RISC - Reduced Instruction Set Computer - ЭВМ с сокращенным набором команд). При этом большое внимание уделяется развитию системных интерфейсов (СИ), составляющих важную компоненту архитектуру мини-ЭВМ.

Современные мультипроцессорные мини-ЭВМ с не - неймановской архитектурой развиваются по двум основным направлениям:

1. распараллеливание по программам;
2. распараллеливание по данным.

В первом случае на процессорах системы одновременно выполняются разные программы, во втором - процессоры ведут идентичную обработку разных частей данных. Наряду с развитием традиционной архитектуры на основе совершенствования элементной базы и нетрадиционных подходов к ней, используется и подход на основе сопроцессоров (векторных, матричных, функциональных, математических и др.). Основные особенности таких сопроцессоров состоят в аппаратной реализации алгоритмов и развитой системе векторных команд над кортежами данных в качестве операндов. Из направлений развития не - неймановской архитектуры мини-ЭВМ можно отметить следующие наиболее перспективные в настоящее время:

- параллельные ВС, ориентированные нба решение задач моделирования больших систем, проведение сложных научно - технических расчетов, прогнозирование и др.;

- единые многофункциональные семейства моделей на основе RISC - подхода, ориентированные на решение сложных технических, технологических, производственных, управленческих и конструкторских задач;

- создание **спецпроцессоров** быстрого ассоциативного доступа к БД/БЗ, позволяющих использовать мини-ЭВМ в качестве эффективных машин баз данных и заданий;

- использование в архитектуре мини-ЭВМ принципов искусственного интеллекта (ИИ) и сопроцессоров векторного типа; при этом расширение внедрения элементов ИИ определяется эффективностью их сопряжения с традиционными архитектурами мини-ЭВМ.

ЭВМ общего назначения

Обладают значительно большими быстродействием и вычислительными возможностями, чем мини-ЭВМ. Доступ к ним строго санкционирован по причине важности хранящейся в них информации, их стоимости и необходимости поддержания специального микроклимата. Основная память ЭВМ колеблется от сотен мегабайт до гигабайт, имея возможности для наращивания; их производительность измеряется

десятками и сотнями миллионов операций в секунду и они могут поддерживать работу с тысячами удаленных терминалов и/или рабочих станций. ЭВМ общего назначения производятся в виде серий совместимых снизу вверх моделей с возрастающими возможностями (от младшей моделей к старшей). По производительности моделей серий ЭВМ принято делить на:

- младшие;
- средние;
- старшие.

Универсальность применения ЭВМ определяет следующие, определяющие этот класс машин, характерные черты: универсальность, совместимость, развитое ПО, агрегатность технических средств при широкой номенклатуре периферийных устройств, высокая технологичность и соответствие широко распространенным мировым стандартам. При этом под универсальностью понимается возможность эффективно решать задачи различных классов и типов из всех областей человеческой деятельности. Совместимость реализуется на аппаратно-программном уровне и предполагает наличие единого системного и прикладного ПО, совместимого снизу вверх для всех моделей одной и той же серии ЭВМ. При этом СПО ЭВМ характеризуется наличием развитых операционных систем, являющихся программным расширением аппаратных средств ЭВМ; обширное прикладное ПО ориентировано на самый широкий круг приложений. Агрегатный принцип организации технических средств, стандартный интерфейс ввода/вывода, позволяющий подключать различные периферийные устройства широкой номенклатуры, совместно с развитым ПО позволяют создавать разнообразные вычислительные комплексы, наиболее отвечающие конкретными приложениям.

Пульт управления и реконфигурации служит для обеспечения интерфейса оператора с ЭВМ, инженерного обслуживания и при наличии мультипроцессорного/многомашинного комплекса для реконфигурации системы. Центральные процессоры обеспечивают непосредственную обработку информации и управление основными устройствами системы. Центральные процессоры работают с ОП, имеющей модульную организацию и хранящей программы, управляющую информацию и информацию для оперативной обработки. Подсистема ввода/вывода включает процессоры ввода/вывода (ПВ/В) и три типа каналов:

1. селекторные (СК);
2. байтмультиплексные (БМК);
3. блок-мультиплексные (БлМК).

Она обеспечивает ввод/вывод информации, осуществляя связь с каналами и ОП под управлением процессоров.

Селекторные каналы работают в монопольном режиме с быстрыми внешними запоминающими (ВЗУ) устройствами. Мультиплексные каналы представляют собой совокупность отдельных логических подканалов двух типов:

- неразделенных;

- разделенных.

Неразделенный подканал обслуживает только одно выделенное ему внешнее устройство (ВУ); при этом несколько неразделенных подканалов могут использовать общие для них ВУ. Разделенный подканал может в режиме разделения времени обслуживать разные ВУ.

Подсистема периферийных устройств выполняет функции хранения и ввода/вывода информации; она включает:

- устройства группового управления (УгУВУ);
- устройства ввода/вывода (УВВ);
- ВЗУ на основе НМЛ и НМД;
- процессорные телеобработки.

На основе стандартной схемы сопряжения: канал УУ ВЗУ к процессору можно подключать несколько каналов указанных типов, к каналу - несколько УгУВУ, а каждое УгУВУ может обслуживать группу ВЗУ. ЭВМ располагает весьма обширной номенклатурой различного типа ВУ. Подсистема телеобработки обеспечивает функции связи с удаленными абонентами и содержит процессоры телеобработки, локальные и удаленные абонентские пункты, а также сети ЭВМ. Развитие архитектуры ЭВМ приводит к созданию виртуальных ЭВМ, когда для пользователя снимаются ограничения на используемые вычислительные ресурсы (процессор, память, ввод/вывод) в разумных пределах и с различными оговорками.

Основным направлением нынешнего состояния и последующего развития архитектуры является интеллектуализация ЭВМ, определяющая эволюцию ЭВМ к системам искусственного интеллекта (ИИ). Новые качества ЭВМ 4-го поколения обеспечивают для пользователя следующие основные возможности:

- общение с ЭВМ без необходимости знания ее устройств и принципов функционирования;
- автоматизация разработки ПО;
- использование БД/БЗ с выходом в сети ЭВМ;
- работа с развитыми экспертными системами.

Предполагается, что ЭВМ 5-го поколения станут ядром распределенных локальных, региональных и глобальным информационно-вычислительных сетей, персональных и коллективных интеллектуальных рабочих станций с доступом ко всему объему накопленных человечеством знаний. Концептуальная архитектура интеллектуальной ЭВМ включает следующие основные подсистемы:

- интеллектуального интерфейса с пользователем (доступ к ЭВМ и взаимодействие в процессе постановки и решения задач, используя естественные формы представления информации и понятия конкретной предметной области);
- анализа и логического вывода (выполнение метафункций выбора метода решения и синтеза программ с учетом контекста и содержания задачи, которые дополняются необходимым знаниями из БЗ; результатом является формирование алгоритма решения задачи);

- решения задач (поддерживается при необходимости проблемно-ориентированными спецпроцессорами);

- управления и обслуживания (обеспечение: надежности системы, перемещаемости программ и данных, автоматического контроля, диагностики и восстановления с целью поддержания живучести системы и других сервисных функций);

- база знаний (накопление, обработка и хранение разнообразных знаний, способствующих и обеспечивающих функциональные возможности ЭВМ интеллектуального характера, а также знание - ориентированных систем).

- Современные ЭВМ в той или иной степени включают в качестве составляющих своих архитектур характерные элементы отмеченных подсистем.

Важной составляющей развития архитектуры ЭВМ, является совершенствование ее системы команд, проводимое по трем основным направлениям. Традиционное направление связано с определением состава и содержания команд, способов адресации и спецификации операндов команд. Здесь четко прослеживается тенденция к увеличению числа команд, допускающих комплексную обработку данных; повышение смыслового уровня операций, обеспечивающих более компактное представление программ и эффективную микропрограммную реализацию команд. Второе направление связано с использованием ограниченных наборов команд (RISC-архитектура) и структур управляемым потоком данных. Третье направление состоит в повышении уровня машинного языка; прежде всего это выражается в развитии описания типов данных. Основной тенденцией в развитии структур ЭВМ общего назначения является разделение функций системы и максимальная специализация подсистем (обрабатывающей, памяти, ввода/вывода, управляющей и обслуживающей) для выполнения данных функций. В русле развития мини- и супер-ЭВМ ЭВМ общего назначения в качестве архитектурных используют многомашинные и мультипроцессорные решения, при необходимости включающие спецпроцессоры, преследующие цели повышений производительности, надежности и живучести ВС.

Фон Нейман сформулировал следующие требования к структуре вычислительной машины.

Так как машина в первую очередь является средством вычислений, она чаще всего будет выполнять элементарные арифметические операции. Следовательно, в ее составе должны быть специализированные органы (organs) для выполнения таких операций; эти органы образуют центральную арифметическую часть машины.

Логическое управление машиной, т. е. управление последовательностью ее операций, должно осуществляться центральным управляющим органом или центральной управляющей частью .

Машина, выполняющая длительную и сложную последовательность операций, должна обладать еще одним важным органом — внутренней памятью достаточно большой емкости, которая могла бы хранить не только

исходные данные, таблицы, промежуточные результаты вычислений и т. д., но и команды, определяющие тип выполняемых операций; иначе говоря — хранить программу вычислений! Отсюда следует важнейший вывод: поскольку команды представляются числовым кодом, над ними можно производить операции как над обычными числами. Таким образом, машина получала возможность модифицировать программу в процессе вычислений.

Машина должна иметь органы хранения информации во внешней записывающей (запоминающей. — Ю. П.) среде , представляющей собой перфокарты, телетайпную ленту, магнитную проволоку и т. д.

Эта информация должна передаваться в центральные арифметическую и управляющую части и в память, а также в обратном направлении с помощью отдельных органов, образующих входную и выходную части машины (предпочтительно, чтобы передача осуществлялась через внутреннюю память, а не через центральную арифметическую часть).

Говоря о принципах выполнения арифметических операций, автор писал о преимуществах двоичной системы счисления и определял размер машинного слова, удовлетворяющего требованиям точности научных вычислений. Кроме того, фон Нейман подчеркивал, что ради экономии оборудования надлежит использовать разрядно-последовательную арифметику, а обсуждая синхронный и асинхронный принципы управления последовательностью действий ЭВМ, отдавал предпочтение первому[4].

Итак, фон Нейман по существу описал структурную схему аналитической машины Бэббиджа, состоящей (в современной терминологии) из арифметического устройства (АУ), устройства управления (УУ), оперативного запоминающего устройства (ОЗУ), внешнего запоминающего устройства (ВЗУ), устройств для ввода и вывода информации (УВВ). Вполне естественными отличиями от идей английского провидца явились рекомендации фон Неймана использовать в устройстве не механическую, а электронную элементную базу и не десятичную, а двоичную систему счисления. Интересно, что если Бэббидж искал аналогии между блоками вычислительной машины и производственными структурными единицами (“мельница”, “склад”), то фон Нейман находил эти аналогии в живом организме (точнее, в нейронных сетях).

ПРОГРАММНАЯ МОДЕЛЬ МИКРОПРОЦЕССОРА

На современном компьютерном рынке наблюдается большое разнообразие различных типов компьютеров. Поэтому возможно предположить возникновение у потребителя вопроса — как оценить возможности конкретного типа (или модели) компьютера и его отличительные особенности от компьютеров других типов (моделей). Рассмотрения для этого одной лишь только структурной схемы компьютера недостаточно, так как она принципиально мало чем различается у разных машин: у всех компьютеров есть оперативная память, процессор, внешние устройства.

К изучению языка ассемблера любого компьютера имеет смысл приступать

только после выяснения того, какая часть компьютера оставлена видимой и доступной для программирования на этом языке. Это так называемая программная модель компьютера, частью которой является ***программная модель микропроцессора***, которая содержит 32 регистра в той или иной мере доступных для использования программистом.

Данные регистры можно разделить на две большие группы:

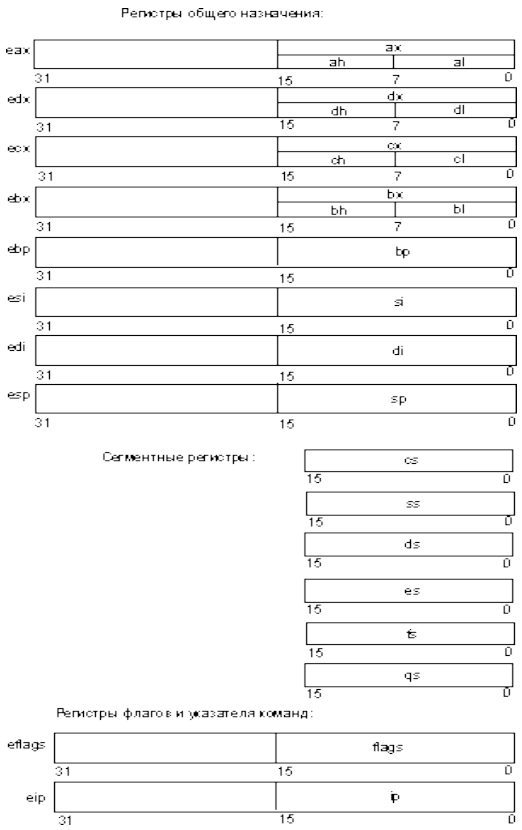
- 16 пользовательских регистров;
- 16 системных регистров.

В программах на языке ассемблера регистры используются очень интенсивно. Большинство регистров имеют определенное функциональное назначение.

Пользовательские регистры

Как следует из названия, *пользовательскими* регистры называются потому, что программист может использовать их при написании своих программ. К этим регистрам относятся (рис. 1):

- восемь 32-битных регистров, которые могут использоваться программистами для хранения данных и адресов (их еще называют регистрами общего назначения (РОН)):
  - **eax/ax/ah/al**;
  - **ebx/bx/bh/bl**;
  - **edx/dx/dh/dl**;
  - **ecx/cx/ch/cl**;
  - **ebp/bp**;
  - **esi/si**;
  - **edi/di**;
  - **esp/sp**.
- шесть регистров сегментов: **cs, ds, ss, es, fs, gs**;
- регистры состояния и управления:
  - регистр флагов **eflags/flags**;
  - регистр указателя команды **eip/ip**.



Пользовательские регистры микропроцессоров i486 и Pentium

Микропроцессоры i486 и Pentium имеют в основном 32-разрядные регистры. Их количество, за исключением сегментных регистров, такое же, как и у i8086, но размерность больше, что и отражено в их обозначениях — они имеют приставку **e** (***Extended***).

Разберемся подробнее с составом и назначением пользовательских регистров.

Регистры общего назначения

Все регистры этой группы позволяют обращаться к своим “младшим” частям (см. рис. 1). Рассматривая этот рисунок, заметьте, что использовать для самостоятельной адресации можно только младшие 16 и 8-битные части этих регистров. Старшие 16 бит этих регистров как самостоятельные объекты недоступны. Это сделано, как мы отметили выше, для совместимости с младшими 16-разрядными моделями микропроцессоров фирмы Intel.

Перечислим регистры, относящиеся к группе регистров общего назначения. Так как эти регистры физически находятся в микропроцессоре внутри арифметико-логического устройства (АЛУ), то их еще называют *регистрами АЛУ*:

- **eax/ax/ah/al** (Accumulator register) — *аккумулятор*. Применяется для хранения промежуточных данных. В некоторых командах использование этого регистра обязательно;
- **ebx/bx/bh/bl** (Base register) — *базовый* регистр. Применяется для хранения базового адреса некоторого объекта в памяти;

- **ecx/cx/ch/cl** (Count register) — *регистр-счетчик*.

Применяется в командах, производящих некоторые повторяющиеся действия. Его использование зачастую неявно и скрыто в алгоритме работы соответствующей команды.

К примеру, команда организации цикла **loop** кроме передачи управления команде, находящейся по некоторому адресу, анализирует и уменьшает на единицу значение регистра *ecx/cx*;

- **edx/dx/dh/dl** (Data register) — регистр *данных*.

Так же, как и регистр *eax/ax/ah/al*, он хранит промежуточные данные. В некоторых командах его использование обязательно; для некоторых команд это происходит неявно.

Следующие два регистра используются для поддержки так называемых цепочечных операций, то есть операций, производящих последовательную обработку цепочек элементов, каждый из которых может иметь длину 32, 16 или 8 бит:

- **esi/si** (Source Index register) — *индекс источника*.

Этот регистр в цепочечных операциях содержит текущий адрес элемента в цепочке-источнике;

- **edi/di** (Destination Index register) — *индекс приемника* (получателя).

Этот регистр в цепочечных операциях содержит текущий адрес в цепочке-приемнике.

В архитектуре микропроцессора на программно-аппаратном уровне поддерживается такая структура данных, как ***стек***. Для работы со стеком в системе команд микропроцессора есть специальные команды, а в программной модели микропроцессора для этого существуют специальные регистры:

- **esp/sp** (Stack Pointer register) — регистр *указателя стека*.

Содержит указатель вершины стека в текущем сегменте стека.

- **ebp/bp** (Base Pointer register) — регистр *указателя базы кадра стека*.

Предназначен для организации произвольного доступа к данным внутри стека.

На самом деле, большинство из регистров могут использоваться при программировании для хранения операндов практически в любых сочетаниях. Но, как мы отметили выше, некоторые команды используют фиксированные регистры для выполнения своих действий. Это нужно обязательно учитывать.

Использование жесткого закрепления регистров для некоторых команд позволяет более компактно кодировать их машинное представление. Знание этих особенностей позволит вам при необходимости хотя бы на несколько байт сэкономить память, занимаемую кодом программы.

Сегментные регистры

В программной модели микропроцессора имеется шесть сегментных регистров: *cs, ss, ds, es, gs, fs*.

Их существование обусловлено спецификой организации и использования оперативной памяти микропроцессорами Intel. Она заключается в том, что микропроцессор аппаратно поддерживает структурную организацию программы в виде трех частей, называемых *сегментами*. Соответственно, такая организация памяти называется ***сегментной***.

Для того чтобы указать на сегменты, к которым программа имеет доступ в конкретный момент времени, и предназначены *сегментные регистры*. Фактически, с небольшой поправкой, как мы увидим далее, в этих регистрах содержатся адреса памяти с которых начинаются соответствующие сегменты. Логика обработки машинной команды построена так, что при выборке команды, доступе к данным программы или к стеку неявно используются адреса во вполне определенных сегментных регистрах. Микропроцессор поддерживает следующие типы сегментов:

1. ***Сегмент кода***. Содержит команды программы.

Для доступа к этому сегменту служит регистр **cs** (code segment register) — *сегментный регистр кода*. Он содержит адрес сегмента с машинными командами, к которому имеет доступ микропроцессор (то есть эти команды загружаются в конвейер микропроцессора).

2. ***Сегмент данных***. Содержит обрабатываемые программой данные.

Для доступа к этому сегменту служит регистр **ds** (data segment register) — *сегментный регистр данных*, который хранит адрес сегмента данных текущей программы.

3. ***Сегмент стека***. Этот сегмент представляет собой область памяти, называемую *стеком*.

Работу со стеком микропроцессор организует по следующему принципу: *последний записанный в эту область элемент выбирается первым*. Для доступа к этому сегменту служит регистр **ss** (stack segment register) — *сегментный регистр стека*, содержащий адрес сегмента стека.

4. ***Дополнительный сегмент данных***.

Неявно алгоритмы выполнения большинства машинных команд предполагают, что обрабатываемые ими данные расположены в сегменте данных, адрес которого находится в сегментном регистре *ds*.

Если программе недостаточно одного сегмента данных, то она имеет возможность использовать еще три дополнительных сегмента данных. Но в отличие от основного сегмента данных, адрес которого содержится в сегментном регистре *ds*, при использовании дополнительных сегментов данных их адреса требуется указывать явно с помощью специальных *префиксов переопределения сегментов* в команде.

Адреса дополнительных сегментов данных должны содержаться в регистрах **es, gs, fs** (extension data segment registers).

Регистры состояния и управления

В микропроцессор включены несколько регистров (см. рис. 1), которые постоянно содержат информацию о состоянии как самого микропроцессора,

так и программы, команды которой в данный момент загружены на конвейер. К этим регистрам относятся:

- регистр флагов **eflags/flags**;
- регистр указателя команды **eip/ip**.

Используя эти регистры, можно получать информацию о результатах выполнения команд и влиять на состояние самого микропроцессора. Рассмотрим подробнее назначение и содержимое этих регистров:

eflags/flags (flag register) — регистр *флагов*. Разрядность *eflags/flags* — 32/16 бит. Отдельные биты данного регистра имеют определенное функциональное назначение и называются флагами. Младшая часть этого регистра полностью аналогична регистру *flags* для i8086. На рис. 2 показано содержимое регистра *eflags*.

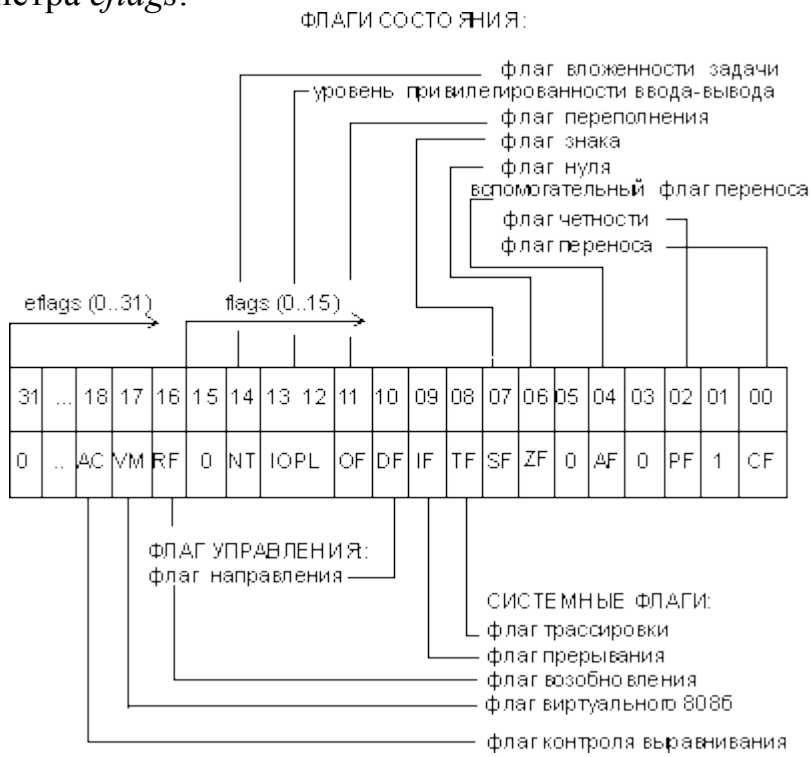


Рис. 2. *Содержимое регистра eflags*

Синтаксис ассемблера

Предложения, составляющие программу, могут представлять собой синтаксическую конструкцию, соответствующую команде, макрокоманде, директиве или комментарию. Для того чтобы транслятор ассемблера мог распознать их, они должны формироваться по определенным синтаксическим правилам. Для этого лучше всего использовать формальное описание синтаксиса языка наподобие правил грамматики. Наиболее распространенные способы подобного описания языка программирования — *синтаксические диаграммы* и *расширенные формы Бэкуса—Наура*. Для практического использования более удобны *синтаксические диаграммы*. К примеру, синтаксис предложений ассемблера можно описать с помощью синтаксических диаграмм, показанных на следующих рисунках.

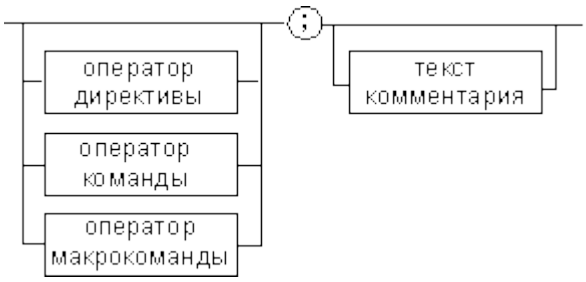


Рис. 1. *Формат предложения ассемблера*

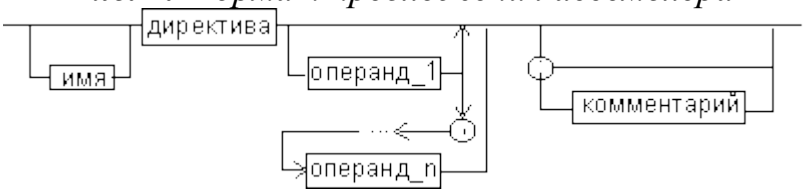
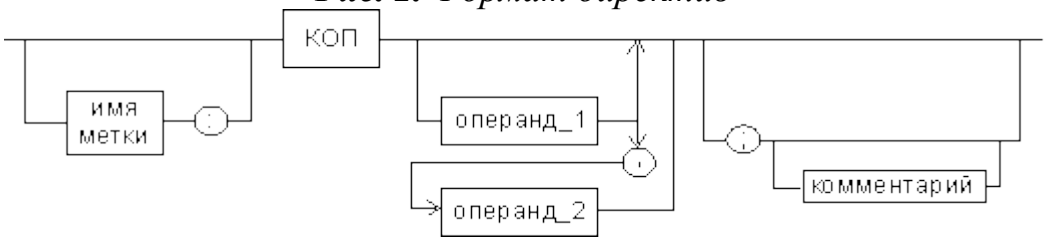


Рис. 2. *Формат директив*



Формат команд и макрокоманд

На этих рисунках:

- *имя метки* — идентификатор, значением которого является адрес первого байта того предложения исходного текста программы, которое он обозначает;
- *имя* — идентификатор, отличающий данную директиву от других одноименных директив. В результате обработки ассемблером определенной директивы этому имени могут быть присвоены определенные характеристики;
- *код операции (КОП) и директива* — это мнемонические обозначения соответствующей машинной команды, макрокоманды или директивы транслятора;
- *операнды* — части команды, макрокоманды или директивы ассемблера, обозначающие объекты, над которыми производятся действия. Операнды ассемблера описываются выражениями с числовыми и текстовыми константами, метками и идентификаторами переменных с использованием знаков операций и некоторых зарезервированных слов.

Как использовать синтаксические диаграммы?

Очень просто: для этого нужно всего лишь найти и затем пройти путь от входа диаграммы (слева) к ее выходу (направо). Если такой путь существует, то предложение или конструкция синтаксически правильны. Если такого пути нет, значит эту конструкцию компилятор не примет. При работе с синтаксическими диаграммами обращайте внимание на направление обхода, указываемое стрелками, так как среди путей могут быть и такие, по которым можно идти справа налево. По сути, синтаксические диаграммы отражают логику работы транслятора при разборе входных предложений программы.

Допустимыми символами при написании текста программ являются:

1. все латинские буквы: **A—Z**, **a—z**. При этом заглавные и строчные буквы считаются эквивалентными;
2. цифры от **0** до **9**;
3. знаки **?**, **@**, **$**, **_**, **&**;
4. разделители **, . [ ] ( ) < > { } + / * % ! ' " ? \ = # ^**.

Предложения ассемблера формируются из *лексем*, представляющих собой синтаксически неразделимые последовательности допустимых символов языка, имеющие смысл для транслятора.

Лексемами являются:

- *идентификаторы* — последовательности допустимых символов, использующиеся для обозначения таких объектов программы, как коды операций, имена переменных и названия меток. Правило записи идентификаторов заключается в следующем: идентификатор может состоять из одного или нескольких символов. В качестве символов можно использовать буквы латинского алфавита, цифры и некоторые специальные знаки — _, ?, $, @. Идентификатор не может начинаться символом цифры. Длина идентификатора может быть до 255 символов, хотя транслятор воспринимает лишь первые 32, а остальные игнорирует. Регулировать длину возможных идентификаторов можно с использованием опции командной строки **mv**. Кроме этого существует возможность указать транслятору на то, чтобы он различал прописные и строчные буквы либо игнорировал их различие (что и делается по умолчанию). Для этого применяются опции командной строки **/mu, /ml, /mx**;
- *цепочки символов* — последовательности символов, заключенные в одинарные или двойные кавычки;
- *целые числа* в одной из следующих систем счисления: *двоичной, десятичной, шестнадцатеричной*. Отождествление чисел при записи их в программах на ассемблере производится по определенным правилам:
  - **Десятичные числа** не требуют для своего отождествления указания каких-либо дополнительных символов, например 25 или 139.
  - Для отождествления в исходном тексте программы **двоичных чисел** необходимо после записи нулей и единиц, входящих в их состав, поставить латинское “**b**”, например 10010101**b**.
  - **Шестнадцатеричные числа** имеют больше условностей при своей записи:
    - *Во-первых*, они состоят из цифр **0...9**, строчных и прописных букв латинского алфавита **a**, **b**, **c**, **d**, **e**, **f** или **A**, **B**, **C**, **D**, **E**, **F**.
    - *Во-вторых*, у транслятора могут возникнуть трудности с распознаванием шестнадцатеричных чисел из-за того, что они могут состоять как из одних цифр 0...9 (например 190845), так и начинаться с буквы латинского алфавита (например **ef15**). Для того чтобы "объяснить" транслятору, что данная лексема не является десятичным числом или идентификатором, программист должен специальным образом выделять шестнадцатеричное число. Для этого на конце последовательности

шестнадцатеричных цифр, составляющих шестнадцатеричное число, записывают латинскую букву “**h**”. Это обязательное условие. Если шестнадцатеричное число начинается с буквы, то перед ним записывается ведущий ноль: **0**ef15**h**.

Таким образом, мы разобрались с тем, как конструируются предложения программы ассемблера. Но это лишь самый поверхностный взгляд.

Практически каждое предложение содержит описание объекта, над которым или при помощи которого выполняется некоторое действие. Эти объекты называются *операндами*. Их можно определить так: ***операнды*** *— это объекты (некоторые значения, регистры или ячейки памяти), на которые действуют инструкции или директивы, либо это объекты, которые определяют или уточняют действие инструкций или директив.*

Система команд микропроцессоров Intel

aaa	aad	aam	aas	adc	add	and
bound	bsf	bsr	bswap	bt	btc	btr
bts	call	cbw	cwde	clc	cld	cli
cmc	cmp	cmps/cmpsb /cmpsw/cmpsd	cmpxchg	cwd	cdq	daa
das	dec	div	enter	hlt	idiv	imul
in	inc	ins/insb /insw/insd	int	into	iret/iretd	jcc
jcxz	jecxz	jmp	lahf	lds	les	lfs
lgs	lss	lea	leave	lgdt	lidt	lods/lodsb /lodsw/lodsd
loop	loope	loopz	loopne	loopnz	mov	movs/movsb /movsw/movsd
movsx	movzx	mul	neg	nop	not	or
out	outs	pop	popa	popad	popf	popfd
push	pusha	pushad	pushf	pushfd	rcl	rcr
rep/repe/repz /repne/repn	ret/retf	rol	ror	sahf	sal	sar

z						
sbb	scas/scasb /scasw/scasd	setcc	sgdt	sidt	shl	shld
shr	shrd	stc	std	sti	stos/stosb /stosw/stosd	sub
test	xadd	xchg	xlat/xlatb	xor		

ИНТЕРФЕЙСЫ СИСТЕМ ВВОДА-ВЫВОДА

Понятие интерфейса и его характеристики

Аппаратным интерфейсом принято называть совокупность правил унифицированного вза-имодействия между отдельными устройствами, а также совокупность аппаратных, програм-мных и конструктивных средств, необходимых для реализации этих правил. Взаимодействие осуществляется с помощью сигналов, передаваемых посредством электрических (или оптичес-ких) цепей, называемых линиями интерфейса; совокупность линий, сгруппированных по фун-кциональному назначению, принято называть шиной интерфейса. Унификация правил взаи-модействия направлена на обеспечение информационной, электрической и конструктивной совместимости; именно унификация и стандартизация лежат в основе построения интерфейсов.

Информационная совместимость достигается за счет единых требований, предъявляемых к структуре и составу линий интерфейса, алгоритмам взаимодействия, способам кодирования и форматам данных, управляющей и адресной информации, временным соотношениям между сигналами.

Электрическая совместимость означает согласованность параметров электрических или оп-тических сигналов, передаваемых средой интерфейса, соответствие логических состояний уров-ням сигналов; электрическая совместимость определяет требования к нагрузочной способности компонентов и характеристикам используемых линий передачи (длина, допустимая активная и реактивная нагрузка, порядок подключения схем согласования и т.д.).

Конструктивная совместимость означает возможность механического соединения электри-ческих цепей, а иногда и механической замены некоторых блоков; этот вид совместимости обеспечивается стандартизацией соединительных элементов (разъемов, штекеров и т.п.), ка-белей, конструкций плат и т.д.

Интерфейсы в СВВ возникают между различными уровнями иерархии физической струк-туры ВС, поэтому требования, предъявляемые к организации обмена, существенно различа-ются. Единый стандартный

интерфейс не смог бы обеспечить эффективную работу разнооб-разных устройств, используемых на различных уровнях иерархии СВВ. Этим объясняется нали-чие системы интерфейсов различных рангов, отличающихся характеристиками и степенью уни-фикации.

В зависимости от требований унификации выделяют:

- физическую реализацию интерфейса, т.е. состав и характеристики линий передачи, кон-струкцию средств их подключения (например, разъем), вид и характеристики сигналов;
- логическую реализацию интерфейса, т.е. протоколы взаимодействия, или алгоритмы фор-мирования сигналов обмена.

В широком смысле протокол определяет совокупность правил реализации определенной функции, например, обмена, и в этом случае может включать требования, охватывающие ин-терфейсы нескольких рангов.

Система аппаратных интерфейсов является одной из основных составляющих понятия ар-хитектуры ВС. На рис.2.1,а и б показаны интерфейсы для машин ЕС ЭВМ и СМ ЭВМ, соот-ветственно. В структуре ВС с выделенными ПВВ отметим интерфейсы четырех рангов. Через интерфейс И1 производится обмен информацией между ОП и процессорами (ЦП или ПВВ); через интерфейс И2—управляющей информацией между ЦП и ПВВ. Интерфейсы И1 и И2 являются внутренними, отражающими особенности конкретной модели и не унифицируются. Интерфейсы ввода-вывода (ИЗ) обеспечивают обмен между ПВВ и контроллерами ПУ (КПУ); они стандартизуются, что дает возможность использовать одинаковые контроллеры и ПУ в различных моделях ЭВМ одной системы.

Для мини- и микро-ЭВМ характерно наличие интерфейса Ио, посредством кото-рого связаны между собой ЦП, ОП и контроллеры. Этот интерфейс принято называть систем-ным (или объединенным), он унифицирован для всего семейства ЭВМ. Контроллеры в мини- и микро-ЭВМ достаточно просты, так как управление обменом между ПУ и ОП осуществляется в значительной мере программным путем. Это позволяет для семейств ЭВМ с различными интер-фейсами Ио использовать одинаковые ПУ (но с разными контроллерами).

Интерфейсы принято характеризовать следующими параметрами:

- видом связи, т.е. возможностью вести дуплексную (сообщения могут одновременно передаваться в двух направлениях, что требует двух каналов связи), полудуплексную сообщения могут передаваться в двух направлениях, но одновременно возможна передача только в одном) или симплексную передачу (сообщения могут передаваться только в одном направлении);
- пропускной способностью, т.е. количеством информации, передаваемой через интерфейс в единицу времени;
- максимально допустимым расстоянием между устройствами или суммарной длиной линий, соединяющих все устройства интерфейса;

- задержками при организации передачи, которые вызваны необходимостью выполнения под-готовительных и завершающих действий по установлению связи между устройствами.

Конкретные значения этих параметров зависят от множества факторов, в частности от информационной ширины интерфейса, способа синхронизации, среды интерфейса, топологической структуры соединений и организации линии интерфейса, совмещения или функционального разделения линий. Все эти факторы определяют организацию интерфейса.

Интерфейс ОШ СМ ЭВМ. Во всех моделях ЭВМ СМ-3, СМ-4 используется унифициро-ванный объединенный интерфейс ОБЩАЯ ШИНА (ОСТ 25-795-78). Он является магистрально-цепочным асинхронным полудуплексным интерфейсом, обеспечивающим возможность парал-лельной передачи 2 байт информации. Передача данных производится между ЦП и ОП, ЦП и ПУ, контроллером прямого доступа к памяти (КПДП) и ОП. В каждый момент времени обмен по магистрали осуществляется только между двумя устройствами, одно из которых является ведущим (или задатчиком ЗДТ), а другое—ведомым (исполнителем ИСП).

С о с т а в л и н и й и о с н о в н ы е о п е р а ц и и . Передача адреса и данных произво-дится по разделенным системам линий, называемым шиной (подшиной) адреса А и шиной (подшиной) данных Д (рис. 2.18). Подшина данных позволяет передавать данные, команды и адреса векторов прерывания. Остальные линии служат для выполнения различных функций по управлению передачами (ШУ1) и занятию ОШ (ШУ2).

Сигнал на линии синхронизации задатчика СХЗ устанавливается ЗДТ и является стробом для сигналов на линиях адреса, данных и У0, У 1. Сброс СХЗ указывает на завершение операции по передаче данных в ЗДТ. Сигнал синхронизации исполнителя на линии СХИ формируется ИСП и является стробом-квитанцией. При операциях чтения установка СХИ означает, что данные помещены ИСП на шину данных, а при операциях записи — что данные приняты ИСП. Сброс СХИ подтверждает, что ИСП получил сброс СХЗ.

Сигнал подготовки ПОДГ выдается ЦП и переводит все устройства, подключенные к ОШ, в исходное состояние. Этот сигнал выдается при нажатии кнопки ПУСК на пульте ЦП, при обна-ружении отказа сети питания, а также при возврате питания в допустимые пределы. Сигналы аварии сети и источника питания на линиях (АСП и АИП) вырабатываются датчиками при нарушении уровней напряжений переменного и постоянного тока. Они позволяют сохранить некоторую информацию в энергонезависимом ОЗУ при аварии в системе питания.

О р г а н и з а ц и я о п е р а ц и й . На средства интерфейса ОШ возлагается предоставление устройствам поочередного права занятие магистрали (арбитраж); установление логической свя-зи между ПУ и программой управления (передача вектора прерывания); передача данных (за-пись и чтение).

О п е р а ц и и ч т е н и я и з а п и с и . В этих операциях участвует ЗДТ и ИСП, причем ЗДТ должен обладать средствами прямого доступа, т.е. автономно формировать текущий адрес данных ИСП.

В случае операции чтения слова (или чтения с паузой) устройство, которое «захватило» ма-гистраль (выставив сигнал ЗАН) и стало задатчиком, выставляет текущий адрес ИСП на линии адреса А[17-00] и код операции чтения (00 или 10) на линии У [0,1]; затем с некоторой задержкой выдает сигнал СХЗ. Этот сигнал в ИСП используется в качестве строба; по его переднему фронту начинается декодирование адреса и команды, затем ИСП выдает данные на шину данных Д [15-00], стробируя их сигналом-квитанцией СХИ. При этом ЗДТ по переднему фронту СХИ принимает данные, сбрасывает сигнал СХЗ, коды адреса и управления. ИСП сбрасывает данные и сигнал СХИ. Если передача завершена, то ЗДТ может освободить магистраль, сняв сигнал ЗАН. Если передача продолжается, то ЗДТ удерживает сигнал ЗАН и вновь выдает код адреса и управления, стробируя их сигналом СХЗ, начиная новый цикл передачи. При выполнении операции записи ЗДТ и ИСП взаимодействуют аналогичным образом.

С р е д а и н т е р ф е й с а . Для всех линий (кроме АИП и АСП) можно использовать стан-дартные усилители-приемники (ПРМ) и усилители-передатчики (ПРД), в которых выход реа-лизован по схеме с открытым коллектором. Уровни сигналов соответствуют ТТЛ-уровням. Согласующие резисторы размещаются на специальных платах, называемых заглушками. Сиг-налы передаются по плоскому кабелю, общая длина каждой линии не должна превышать 15 м, а число ПРМ и ПРД на одной линии не должно превышать 20.

Интерфейс МПИ. Этот интерфейс (ГОСТ 26765.51-86) представляет собой модификацию интерфейса ОШ и используется во многих микро-ЭВМ, например, серии «Электроника-60» МПИ является магистрально-цепочным асинхронным параллельным полудуплексным интер-фейсом с совмещенной шиной для передачи адреса и данных. В МПИ используются как одно-, так и двунаправленные линии. Передача адреса и данных по линиям АД [15-00] магистрали осуществляется последовательно. В МПИ предусмотрено пять уровней приоритетов ПУ, одна-ко обязательными являются только два: в ы с ш и й - для прямого доступа в память; н и з ш и й— для программного обмена. Аналогично ОШ приоритет устройства определяется его расположением на линии разрешения (прямого доступа или передачи) относительно арбитра.

Малые интерфейсы периферийных устройств

Группа малых интерфейсов (ранга И4) обеспечивает подключение ПУ к контроллерам; требования, предъявляемые к малым интерфейсам, могут существенно различаться в зависи-мости от особенностей ПУ. Интерфейсы ПУ со специализированными контроллерами не уни-фицируют; контроллеры конструктивно объединяют с самим ПУ; при этом устройство подклю-чается непосредственно к системному интерфейсу. Если же контроллер

предназначен для уп-равления несколькими ПУ, то малый интерфейс унифицируют, что позволяет уменьшить но-менклатуру контроллеров (посредством контроллеров одного типа можно подключать к ЭВМ различные типы ПУ) и использовать одни и те же ПУ в различных типах ВС. Функции управ-ления ПУ разбиваются на два уровня — непосредственного управления механизмами и аппа-ратурой ПУ, осуществляемого схемами местного управления, и преобразования алгоритмов обмена системного и малого интерфейсов, реализуемого контроллером. Наиболее характер-ными примерами малых интерфейсов могут служить интерфейсы накопителей на магнитных дисках (НМД) для подключения к групповым контроллерам, интерфейсы параллельный ИРПР и последовательный ИРПС для подключения дисплеев, печатающих устройств, а также интер-фейсы для подключения терминалов (стыки). Для унификации контроллеров малых ЭВМ все чаще используют «системный интерфейс малых ЭВМ» (SCSI), предназначенный для подклю-чения основных типов ПУ. Малые интерфейсы во многих случаях должны обеспечивать уда-ление ПУ на значительные расстояния. Ниже рассмотрены наиболее распространенные малые интерфейсы, обладающие наибольшей степенью унификации — ИРПР, ИРПС и С2.

ИРПР - параллельный, радиальный, асинхронный симплексный интерфейс - служит для подключения сравнительно медленных ПУ к контроллеру. Интерфейс унифицирован физически и имеет несколько модификаций логической организации. Линии интерфейса являются одно-направленными и связывают один приемник (П) и один передатчик-источник (И). Функции приемника и источника могут выполняться как контроллером, так и ПУ. Интерфейс обеспе-чивает возможность параллельной передачи не более 16 бит.

НОСИТЕЛИ ИНФОРМАЦИИ

Классификация и характеристики носителей информации.

Носитель информации - физическое тело, используемое при записи для сохранения в нем или на его поверхности сигналов информации (ГОСТ 13699-80). В процессе записи производится преобразование сигналов информации в пространственное изменение состояния или формы носителя информации с целью сохранения и последующего воспроизведения записанной информации. Воспроизведение информации - процесс получения записанной информации от но-сителя в любой сигнальной форме.

В практике получил распространение термин «машинные носители», т. е. носители, нформация с которых может быть воспроизведена с помощью относительно простых технических средств и которые являются унифицированными, легкотранспортируемыми и обмениваемыми документа-ми. К таким носителям относятся магнитные ленты, съемные магнитные и оптические диски.

Информация, записанная на носитель, может иметь привычную для человека форму (символы, графическое изображение) или быть

закодированной. Видоизменение носителя в процессе записи может сводиться к изменению формы (например, перфорации), нанесению контрастного изображения (например, при печати или записи на оптический диск), созданию различных неоднородностей на носителе (магнитной — в носителях с магнитным слоем, электрической—при записи электростатическим или электрофотографическим способом). Некоторые носители (бумажные бланки) предназначены для записи информации как ручным, так и машинным спосо-бом. Однако для большинства носителей для записи и воспроизведения информации используются специальные технические средства—периферийные устройства.

По конструктивному исполнению различают носители - ленты, карты, диски (реже жетоны, проволоку).

По характеру использования носители делят на однократного и многократного пользования (последние допускают многократную перезапись информации).

Важнейшая характеристика носителей—линейная, поверхностная и объемная плотности записи информации, которые непосредственно определяют скорости записи и воспроизведения, а также стоимость хранения единицы (например, бита) информации.

Алфавит символовНабор символов, которые могут быть использованы при обмене и обработке алфавитноцифровой информации в АСУ, языках программирования, различных ПУ, включая и устройства передачи данных, определяется ГОСТ 19767—74 (СТ СЭВ 359—76). Стандартный набор графических символов, т. е. символов, которые могут быть получены ручным или печатным способом или воспроизведены визуально на устройстве отображения информации, включает прописные и строчные латинские и русские (кириллические) буквы, арабские цифры, знаки препинания, арифметических и логических операций и другие специальные знаки, служебные (управляющие) символы - символы, появление которых вызывает некоторое действие, влияющее на процесс регистрации, передачи, обработки или интерпретации данных.

Служебные символы подразделяются на символы связи, формата, устройства, разделители информации и расширения. К символам связи относятся, например, символы «Начало заголовка» (НЗ), «Начало текста» (НТ), «Конец текста» (КТ), «Конец передачи» (КП). Смысл этих символов подчеркивается их названием: они указывают на «границы» соответствующих смысловых частей передаваемой информации. К этому же подклассу служебных символов относятся и символы, служащие для установления связи между абонентами. Например символ «Кто там?» (КТМ), «Подтверждение» (ДА), «Отрицание» (НЕТ). Символ КТМ используется для запроса от удаленной станции. Ответом может служить название станции или информация о ее состоянии.

Символы формата используются для управления печатающим (отображающим) устройством в целях получения формата данных

необходимого вида. К ним относятся «Возврат каретки» (ВК), «Возврат на шаг» (ВШ), «Перевод строки» (ПС), «Горизонтальная табуляция» (ГТ) и др.

Символы устройства предназначены для включения или выключения вспомогательного оборудования в линиях передачи информации. Символы—разделители информации—служат для иерархического разделения информации в массиве на элементы, записи, группы, файлы.

Кодирование текстовой информации

Текстовая информация представляется последовательностью алфавитно-цифровых символов, каждый из которых определенным образом кодируется. Для кодирования символов в качестве внутреннего кода ЭВМ наиболее часто используется двоичный код обработки информации (ДКОИ), построенный на основе международного кода EBCDIC. Этот код позволяет кодировать 256 символов, при этом кодирование десятичных цифр и букв строится по весовому принципу, согласно которому двоичный код символа (или его «вес») возрастает последовательно на единицу при переходе от цифры к цифре (в порядке возрастания цифр от 0 до 9) и от буквы к букве (в алфавитном порядке). Такое построение кода значительно упрощает наиболее частые операции обработки текстов — сортировку и поиск. Кодирование символов посредством ДКОИ осуществляется согласно табл. 4.1. Символы располагаются в узлах таблицы на пересечении строк и столбцов; строки и столбцы пронумерованы. Код символа определяется путем приписывания к номеру столбца номера строки, на пересечении которых находится символ, например, символу W соответствует код 11100110. Для сокращения записи кода часто используется шестнадцатиричная система; символу W при этом соответствует обозначение E6.

Для кодирования алфавитно-цифровых символов в ПУ, в частности в устройствах, используемых при передаче дискретных сообщений по каналам связи, хранении информации в системах внешней памяти, при обмене с другими машинами, в клавиатурах, устройствах печати и т.п., наиболее распространен 7-разрядный код КОИ-7 (ГОСТ 13052-74). При выборе системы кодирования определяющим здесь являются требования стандартизации, обеспечивающие совместимость различного оборудования. Весь алфавит графических и служебных символов распределен по трем кодовым таблицам: КОИ-7Н0, КОИ-7Н1 и КОИ-7С1 (табл.4.2). Каждая таблица КОИ-7НО и КОИ-7Н1 содержит по 128 графических и служебных символов. Таблица КОИ-7НО содержит буквы латинского, а таблица КОИ-7Н1—русского алфавитов; цифры, графические знаки и служебные символы в обеих таблицах повторяются. Семиразрядный код символа определяется приписыванием номера строки к номеру столбца, на пересечении которых он находится. Так, букве V соответствует код 101 0110 (см. табл. КОИ-7Н0); такой же код соответствует букве «Ж» (см. табл. КОИ-7Н1). Таблица КОИ-7С1 содержит только управляющие символы (32 символа), которые кодируются аналогично; так, указанному коду 1010110 соответствует символ «ВП» (при приеме этого кода в ПУ формируется сигнал, используемый для

перевода печатающего механизма в верхнее положение). Для обеспечения однозначности кодирования предусмотрены специальные управляющие символы ВХ, ВЫХ, АР2, называемые символами переключения регистров и служащие для увеличения мощности алфавита. Если в принимаемой последовательности симво-лов встречается символ «ВХ» (код 0001111), то все последующие коды расшифровываются в соответствии с табл. КОИ-7Н0; если встречается символ «ВЫХ», то все последующие коды расшифровываются в соответствии с табл. КОИ-7Н1. Для расшифровки кода символа в соответствии с табл. КОИ-7С1 каждому такому коду должен предшествовать код 0001011 символа АР2. При отсутствии управляющих символов переключения регистров кодирование и декодирование осуществляется согласно табл. КОИ-7Н0. Чтобы сократить число символов переключения регистров в сообщении, алфавит разбивается на группы символов, включаемых в одну таблицу, с учетом вероятности их совместного использования.

Оптические диски

Середина 80-х годов ознаменовалась введением в практику оптических ЗУ, носителем информации в которых является оптический диск (ОД). Информация на ОД представляется участками на дорожках—концентрических окружностях с неоднородной отражающей способностью, что позволяет относительно просто воспроизводить ее с помощью лазерных уст-ройств.

Плотность записи информации на ОД в десятки раз превышает плотность записи на МД. Поэтому ЗУ на ОД отличаются большой емкостью и малой стоимостью хранения одного бита.

- На стеклянном диске с металлическим покрытием, выпускаемом фирмой «Hitachi Ltd» (Япония), на одной поверхности хранится 1310 Мбайт (на поверхности ОД 41300 дорожек, поперечная плотность 640 дорожек/мм). Такой объем информации примерно соответствует 700000 страниц текста формата А4.

Основной недостаток таких ЗУ—большое время поиска информации на ОД (по сравнению с ЗУ на МД) из-за невозможности достижения высокой скорости перемещения массивной оптической головки для записи и считывания при большой поперечной плотности дорожек (точность позиционирования 0,1 мкм). В первую очередь стали использоваться ОД небольшого размера (компакт-диски диаметром 127 мм) с записанными на них постоянными данными, а также ОД с возможностью однократной записи.

Повышенный интерес вызывают у пользователей диски с возможностью перезаписи информации. На одном из опытных ОД при записи лазерным лучом материал из кристаллического состояния с высоким коэффициентом отражения переходит в аморфное состояние с низким коэффициентом отражения. В процессе стирания—обратный переход. Допускается до 1 млн. циклов перезаписи. Исследуются ОД, в которых при записи и считывании информации используются магнитооптические явления.

Назначение, классификация и основные характеристики устройств отображения информации (УОИ).

УОИ предназначены для визуализации текстовой и графической информации без её долговременной фиксации.

Все УОИ делят на алфавитно-цифровые и графические. УОИ различаются также типом используемого индикатора и способом формирования изображения на экране.

Преобразование электрических сигналов в изображение основывается на явлениях люминесценции, газового разряда, изменения оптических свойств жидких кристаллов, светоизлучения полупроводниковыми материалами и т. п.

Технические характеристики:

– разрешающая способность;

– качество воспроизводимых цветов или градаций яркости;

– размер экрана (диагональ);

– масса и габариты;

– частота сканирования — вертикальная и горизонтальная развёртка (для ЭЛТ).

Дисплеи на базе ЭЛТ. Методы построения изображений на экране ЭЛТ. Режимы работы и их особенности.

ЭЛТ представляет собой стеклянную колбу с плоским основанием-экраном, покрытым с внутренней стороны люминофором. В колбе размещено несколько электродов. Катод 1 нагревается подогревателем 6 до температуры около 1100 К. Под воздействием электростатического поля ускорения, создаваемого катодом и анодом 2, электроны с поверхности нагретого катода устремляются в сторону анода и экрана 3. Эти электроны проходят через фокусирующую систему 5 и образуют узкий пучок, который, ударяясь о поверхность экрана, вызывает вторичную эмиссию и свечение люминофора. Пучок электронов отклоняется электростатической или электромагнитной отклоняющей системой 4.

Графические изображения могут формироваться двумя способами.

В векторном дисплее электронный луч непрерывно «вырисовывает» контур изображения. Само изображение формируется из отдельных элементарных отрезков (векторов).

В растровых дисплеях изображение получается с помощью матрицы точек, которые могут «светиться», а могут быть невидимыми: электронный луч пробегает по строкам экрана, подсвечивая требуемые точки (зёрна) люминофора.

7. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

ЛАБОРАТОРНАЯ РАБОТА № 1

РАЗРАБОТА ПРОГРАММ НА ЯЗЫКЕ АССЕМБЛЕРА ДЛЯ СЕМЕЙСТВА КОМПЬЮТЕРОВ СОВМЕСТНЫХ С IBM PC.

ЦЕЛЬ: показать основные требования к программам на языке ассемблера, модели памяти и структуры программ, этапы ассемблирования, компоновки и выполнения программы.

1. Введение.

Язык ассемблера является довольно необычным и сложным, это объясняется сегментной организацией памяти и одновре менной адресацией четырех сегментов. В языке имеется более 100 базовых символических команд, в соответствии с которыми ассемблер генерирует более 3800 машинных команд. Кроме того, в распоряжении программиста имеется около 20 директив, предназначенных для определения памяти, инициализации переменных и т.д. Учитывая, что в любой программе имеется множество данных и адресов, то разработка ассемблерных программ превращается в кропотливый труд и требующий много усилий.

Тем не менее затраченное время и усилия на изучение языка ассемблер вознаградится. Ассемблерные программы обычно эффективнее (занимает меньший объем памяти и выполняются быстрее) эквивалентных программ, транслируемых с тех или иных языков высокого уровня.

2. Организация программного обеспечения.

Программное обеспечение для микрокомпьютеров создаются на языках высокого уровня, таких как Бейсик, Фортран, Паскаль, Си и др. Однако в тех случая, когда требуется понимание основных принципов работы компьютера, полная реализация всех его возможностей, использование внешних устройств с наибольшей эффективностью - необходимо программирование на уровне машинных языков (кодов). Но на машинном языке программы не кодируют. Для этого используется близкий к машинному языку - язык ассемблера, в котором программист пользуется символическими мнемокодами вместо машинных команд и описательными именами для полей данных и адресов памяти.

При программировании на языке ассемблера первое,что должно быть известно об аппаратном обеспечении микрокомпьютера, это какие регистры он имеет и как они используются при выполнении машинных команд, а также какие существуют способы адресации при выборке команд и данных из основной памяти и т.д. Указанные вопросы рассматриваются на процессоре 8086/88 как типичного и описывается механизм сегментации памяти, являющейся реализацией одной из современных идей организации памяти.

В процессоре 8086/88 основная (оперативная) память разделена на участки по 1 байту (8 бит каждый), которые различаются номерами 0, 1, 2, 3 и т.д.

Это позваляет выбрать, например, команду длиной 1 байт, занимающей участок памяти под номером 0, или выбор данного, имеющего длину 4 байта и занимающего область памяти, включающую участки с номерами 1, 2, 3, 4.

Возможен и другой способ адресации основной памяти, при котором память разделяется на области, называемые сегментами (рис.1 б)). При объединении нескольких однобитных участков памяти в один сегмент и использовании им как единым целым, появляется возможность выделить в памяти область программ и область данных и использовать каждую из них независимо. Оказалось, что оперировать сегментом очень удобно, поскольку сегмент может быть выбран в любом месте основной памяти и размер его также может быть любым.

На рис.1, в качестве примера, память распределена следующим образом: в 1-м сегменте - область стека, во 2-м - область данных, в 3-м - дополнительная область данных, в 4-м – область программ (кодов). Сегментные регистры CS, SS, DS, ES указывают, с какого адреса начинается область памяти, занимаемая каждым из перечисленных выше сегментов основной памяти. В частности, регистр CS (сегмент кодов), используется для указания, в каком сегменте находится программа. IP (указатель адреса очередного кода программы), SP (указатель стека), BP (указатель базы) представляют собой регистры, указывающие адреса команд и данных в соответствующих сегментах (рис.1 б)). Для вычисления исполнительного адреса данных необходимо знать номер сегмента и адрес внутри сегмента (номер байта, называемый также смещением).

Из элементов, составляющих микропроцессор, набор регистров - это главное, что нужно знать при написании программы

Регистр IP (указатель команд), указывающий адрес команды, подлежащей выполнению на следующем шаге, устанавливается автоматически средствами микропроцессора и не может изменять свое значение под действием команды, входящей в состав программы, т.е. с точки зрения программы он не существует,
и поэтому на рис.2 не показан.

Регистры AX, BX, CX, DX предназначены для обработки данных с помощью команд. Эти регистры называются регистрами общего назначения, т.к. допускают произвольное использование (могут содержать и данные и адреса). Каждый из них имеет длину 16
разрядов, но могут быть разделены на левую (старшую) и правую (младшую) половину, которые могут быть использованы как соответствующие самостоятельные регистры длиной 8 разрядов. В этом случае регистры получают имена AH, AL, BH, BL, CH, CL, DH, DL. Буквы A, B, C, D, входящие в эти названия, являются начальными буквами слов "аккумулятор", "база", "счетчик", "данные" соответсвенно, отражающих

их основное назначение, однако они могут использоваться для совершенно произвольных операций. Так регистру AX(AH,AL) присущи некоторые функции, не свойственные другим регистрам, и поэтому он является регистром общего назначения с расширенными функциями (аккумулятором). Для 4-х регистров SI, DI, BP, SP, имеющих длину 16 разрядов (2 байт) основными операциями являются операции, соответствующие их названиям, но в общем случае они могут использоваться в разных целях, и в часности для хранения 16-разрядных адресов.

В нижней части рис.2 показан регистр флажков. Значения его битов устанавливается в зависимости от результатов выполнения команды АЛУ, например, от результата арифметической операции (положительный, отрицательный или равный нулю). Содержимое этого регистра используется для определения дальнейшего порядка выполнения программы. Подробное рассмотрение регистра флажков в нашу цель не входит.

Остальные регистры: SS, DS, ES, BS - содержат начальные адреса сегментов. Все эти 4-е регистра 16-разрядные, но их содержимое смещено на четыре разряда влево по отношению к другим регистрам. Это связано со следующим: область памяти, которая начинается на границе параграфа, т.е. по любому адресу, кратному 16, называется сегментом. В регистре содержится 16 бит. Т.к. адрес сегмента всегда на границе параграфа, младшие четыре бита адреса равны нулю. FFFOH позволяет адресовать до 65520 (плюс смещение) байт. Поэтому адрес хранится в сегментном регистре как щестнадцатиричное nnnnH, а процессор полагает, что имеется еще четыре нулевых младших бита (одна шестнадцатиричная цифра), т.е. nnnnOH. Таким образом, FFFFOH позволяет адресовать до 1048560 байт. Процессор 80286 использует 24 бита для адресации так, что FFFFFO позваляет адресовать до 16 млн. байт, а процесор 80386 может адресовать до 4 млрд. байт.

Минимальный размер сегмента составляет 16 байт, максимальный размер составляет 64 Кбайт. Любой сегмент можно помещать в любое место основной памяти - свойство перемещаемости.

3. Структура и образ памяти программы .EXE.

Совокупность команд и директив, представляемая как функциональная единица для обработки ассемблером, называется исходным модулем. Небольшая программа может состоять из одного модуля, а сложная из десятков. Рассмотрим минимум синтаксических правил языка ассемблера, необходимых для написания простой программы.

1) Первой строкой программы на языке ассемблера является директива NAME (наименовать), которое присваивает внутреннее имя объектному модулю, генерируемому ассемблеру:

```
NAME   имя_модуля ;
```

Почти неиспользуемая, иначе, в первой строке можно использовать директиву TITLE
(заголовок) с именем программы.

2) Программа пишется в виде одного сегмента кодов. Для этого второй строкой программы должно быть предложение, состоящее из имени программы и директивы SEGMENT (сегмент). Число пробелов между словами (именем и директивой) может быть любым. После этой директивы должно быть записана директива ASSUME (считать, предположить), которая устанавливает для ассемблера соответствие между конкретными сегментами и сегментными регистрами. При использовании данных, обозначаемые обычно латинскими буквами, может быть точно также указан и DS (сегмент данных) и SS(сегмент стека). Заканчивается сегмент директивой ENDS.

```
имя_программы   SEGMENT
                ASSUME  CS:имя_программы
              . . .
имя_программы   ENDS
```

3) Сегмент кодов обычно содержит основную программу и несколько подпрограмм, и для каждой из них записывается название, после названия в начале подпрограммы пишется директива PROC, а в конце - ENDP.

```
имя_программы   PROC
                .
                .
                .
имя_программы   ENDP
```

4) Запись BX означает, что данные размещены в регистре BX; запись [BX] означает, что данные расположены по адресу, находящемуся в регистре BX. Это косвенная адресация, которая реализуется только с помощью регистров BX, BP, SI, DI. Другие
регистры для этой цели использовать нельзя.

5) Для инициализации ассемблерной EXE - программы система DOS имеет четыре требования:

1. Указать ассемблеру, какие сегментные регистры должны соответствовать сегментам (используется директива ASSUME);
2. Сохранить в стеке адрес, находящийся в регистре DS, когда программа начнет выполнение (команда PUSH заносит значение в стек);
3. Записать в стек нулевой адрес (точнее, смещение);
4. Загрузить в регистр DS адрес сегмента данных, т.к. загрузчик DOS устанавливает правильные адреса стека в регистре SS и сегмента кодов в регистре CS, а регистр DS использует для других целей, поэтому его необходимо инициализировать.

6) Начало комментария к программе отмечается точкой с запятой. Действие комметария прекращаетсся в конце строки. Поскольку комментарии в ассемблере в процессе трансляции игнорируются, после точки с запятой можно записать любые замечания.

В языке ассемблера вводится понятие логического сегмента, который просто означает "часть программы". Логические сегменты отражают разработку программы в виде отличающихся друг от друга областей кода (собственно программы), данных и стека.

Простые программы могут состоять всего из трех логических сегментов: один для машинного кода, один для переменных (данных) и один для стека. Логические сегменты как средство языка ассемблера взаимосвязаны с физическими сегментами,

представляющими собой особенности архитектуры микропроцессора.

Каждый логический сегмент в ассемблерной программе определяет наименованную область памяти, которая адресуется с неизменным содержимым одного из сегменых регистров. Логический сегмент должен начинаться с директивы SEGMENT и заканчиваться директивой ENDS, этим двум директивам присваивают имя, определенное программистом. Рассмотрим исходную программу с четырьмя логическими сегментами: стековый сегмент с символическим именем STACK_SEG, сегмент данных - DATA_SEG, дополнительный сегмент - EXTRA_SEG, командный сегмент - CODE.

```
                NAME EXAMPLE
    Title       Программа типа .EXE
;---------------------stack segment-------------------------

    STACK_SEG    SEGMENT  Stack 'Stack'
                DW   32 DUP(?); Зарезервировать 32 слова
    STK_TOP      LABEL   WORD  ; Вершина стека
    STACK_SEG    ENDS

;---------------------data segment--------------------------

    DATA_SEG     SEGMENT
       VAR_1   DW    0; Определить и инициализдировать
       VAR_2   DW    0; две переменные
    DATA_SEG     ENDS

;---------------------extra segment-------------------------

     EXTRA_SEG    SEGMENT
          V      DB   2000 DUP(0)
     EXTRA_SEG    ENDS
```

```
;--------------------code segment-------------------------

CODE_SEG    SEGMENT   'Code'
START       PROC  FAR
            ASSUME  CS:CODE_SEG,DS:DATA_SEG,
                 ES:EXTRA_SEG,SS:STACK_SEG
            PUSH  DS          ; записать DS в стек
            SUB  AX,AX        ; установить 0 в AX
            PUSH  AX          ; записать 0 в стек
            MOV   AX,DATA_SEG  ; занести адрес
            MOV   DS,AX       ; DATA_SEG в DS
            MOV   AX,EXTRA_SEG ; занести адрес
            MOV   ES,AX       ; EXTRA_SEG в AX

; Здесь продолжается запись команд программы

            RET               ; возврат в DOS
START       ENDP
CODE_SEG    ENDS
            END    START
```

В первой строке программы находится директива NAME (имя программы). Директива Title задает программе текстовый заголовок, который будет выводится на все страницы листинга трансляции. Программа содержит четыре логических сегмента. Переменные, константы, таблицы и другие данные группируются в сегменте данных. Здесь в сегменте данных DATA_SEG содержится всего две переменные VAR_1 и VAR_2, которые определены и инициализированы (установлены в нуль) с помощью директив DW, что определяет тип WORD (слово), поэтому в области данных имеется два слова.

Стековый сегмент используется для хранения промежуточных данных, связей процедур. Строка описания сегмента стека должна содержать класс сегмента - 'STACK', а также тип объединения - STACK. Тип объединения указывает компоновщику, каким образом должны объединятся одноименные сегменты разных модулей - накладываясь друг на друга (тип объединения COMMON) или присоединяясь друг к другу (тип объединения STACK для сегментов стека или PUBLIC для всех остальных сегментов). Хотя для одномодуль ных программ тип объединения значения не имеет, для сегмента
стека обязательно указание типа STACK, поскольку в этом случае при загрузке программы выполняется автоматическая инициализа ция регистров SS (адресом начала сегмента стека) и SP (смещением конца сегмента стека).

Первая строка в сегменте STACK_SEG содержит директиву DW с операндом 32 DUP(?), которая резервирует 32 слова памяти, но не инициализирует их.Для EXE-программ для стекаобеспечивается не менее 32 слов. Конструкция 32 DUP(?) означает - " выдать 32 копии значения

находящихся в круглых скобках". Вопросительный знак указывает, что значения резервируемых слов произвольны и никакие предположения о них недопустимы. Следующая строка в сегменте стека содержит директиву LABEL (отметить): STK_TOP LABEL WORD.

Данная директива определяет имя STK_TOP, которое относится к слову, находящимуся после 32 слов, т.е. имя STK_TOP идентифицирует вершину сегмента стека STAK_SEG пока пустого. Значения смещения STK_TOP от начала сегмента находится в указателе стека SP, когда стек пустой. Содержимое SP будет уменьшаться на 2 при выполнении каждой команды, включающей слово в стек. Когда стек полностью заполнен (в нем находится 32 слова), содержимое SP равно нулю. Включение еще одного слова в заполненный стек приведет к неуправляемому поведению системы.

Третий сегмент, который рассматривается как дополнительный сегмент программы EXTRA_SEG, используется для заполнения выходных данных.

Слово 'CODE', стоящее в апострофах в строке описания программного сегмента, указывает класс сегмента - "программный". Классы сегментов анализируются компоновщиком и используются им при компоновке загрузочного модуля: сегменты, принадлежащие одному классу, загружаются в память рядом. Для простых программ, включающих один программный сегмент и один сегмент данных, эта процедура не имеет значение, однако для правильной работы компоновщика при описании программного сегмента необходимо указание его класса 'CODE'.

Сегмент кода содержит директивы PROC и ENDP – предназначены для указания начала и окончания процедуры. Вместе с директивой PROC могут быть заданы параметры FAR и NEAR, которые позволяют ассемблеру следить за тем, является ли процедура межсегментной или внутрисегментной соответственно.

Директива ASSUME используется для указания, какому сегменту принадлежит тот или иной сегментный регистр. Определение CS:CODE_SEG указывает транслятору, что данный сегмент является программным и будет адресоваться с помощью сегментного регистра CS. Определение DS:DATA_SEG закрепляется за сегментом DATA_SEG сегментный регистр DS, как регистр, используемый по умолчанию, что позволяет ссылаться на переменные, описанные в сегменте DATA_SEG, без явного указания регистра DS. При этом ассемблер проверяет, действительно ли они описаны в сегменте DATA_SEG. Определение SS:STECK_SEG в простых программах, не осуществля-
ющих операции с данными в стеке, и ES:EXTRA_SEG не обязательно.

Далее команды PUSH, SUB и PUSH инициализируют стек, следующие две команды MOV обеспечивают адресацию сегмента данных, аналогично, проходит адресация дополнительного сегмента. Поскольку передача в сегментные регистры непосредственных значений не допускается, в качестве "перевалочного пункта" используется регистр AX. После того, как регистр DS инициализирован, программа может обращаться к данным, описанным в регистре данных.

Еще раз отметим, что регистры SS и CS инициализируются автоматически при загрузке программы для выполнения, а инициализация регистра DS и, если требуется, ES лежит полностью на самой программе.

Все программы, использующие стандартную инициализацию стека и сегмента данных, могут использовать эти команды, записанные в файл-заготовку. И для каждой новой программы копировать этот файл с новым именем, и, используя затем, редактор, записать дополнительные команды.

Последняя строка программы с директивой END, имеющей операнд START, сообщает ассемблеру о достижении конца исходного модуля и необходимость начать выполнение программы с директивы (команды),отмеченной именем (меткой) START (точка входа в программу).

При загрузке программы сегменты размещаются в памяти, как показано на рис.3.

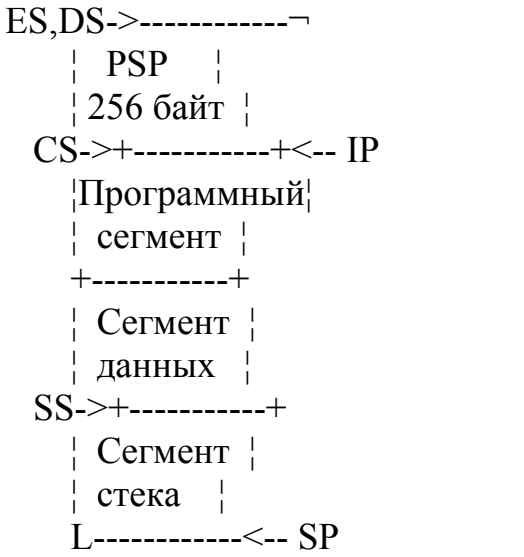


Рис.3. Образ памяти программы .EXE

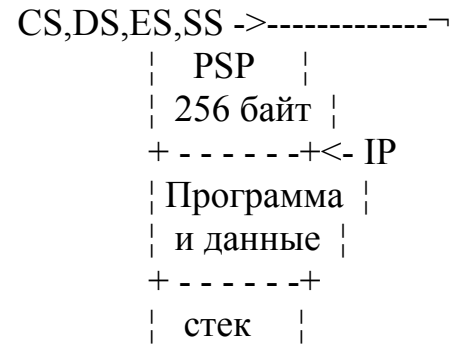
Образ программы в памяти начинается с префикса программного сегмента (Program Segment Prefics, PSP), образуемого и заполняемого системой. PSP всегда имеет размер 256 (100h) байт и содержит таблицы и поля данных, используемые системой в процессе выполнения программы. Вслед за PSP располагаются сегменты программы. Сегментные регистры автоматически инициализируются следующим образом: ES и DS указывают на начало PSP (что дает возможность, сохранив их содержимое, обращаться затем в программе к PSP), CS - на начало программного сегмента, а SS - на начало сегмента стека. В указатель команд IP загружается относительный адрес точки входа в программу (из операнда директивы END), а в указатель стека SP - смещение конца сегмента стека. Таким образом, после загрузки программы в память адресуемыми оказываются все сегменты, кроме сегмента данных. Инициализация регистра DS в первых строках программы позволяет сделать адресуемым и этот сегмент.

4. Структура и образ памяти программы .COM.

Программа типа .COM отличается от программы типа .EXE тем, что содержит лишь один сегмент, включающий все компоненты программы: PSP, программный код (т.е. оттранслированный в машинные коды программные строки), данные и стек. Структура типичной программы типа .COM на языке ассемблера выглядит следующим образом:

```
        title   Программы типа .COM
text    segment 'code'
        assume  CS:text,DS:text,ES:text,SS:text
        org     100h
myproc  proc
        ...                     ;Текст программы
myproc  endp
        ...                     ;Определение данных
text    ends
        end     myproc
```

Программа содержит единственный сегмент text, которому присвоен класс 'CODE'. В директиве ASSUME указано, что все четыре сегментных регистра будут указывать на этот единственный сегмент. Как и в программе типа .EXE, определения ES:text и SS:text
не обязательны. Оператор ORG 100h резервирует 256 байт для PSP. Заполнять PSP будет по-прежнему система, но место под него в начале сегмента должен отвести программист. В программе нет необходимости инициализировать сегментный регистр DS, поскольку
его, как и остальные сегментные регистры , инициализирует система. Данные можно разместить после после программной процедуры, или внутри нее, или даже перед ней. Следует только иметь в виду, что при загрузке программы типа .COM регистр IP всегда
инициализируется числом 100h, поэтому сразу вслед за оператором ORG 100h должна стоять первая выполнимая строка строка программы. Если данные расположены в начале программы, то перед ними следует поместить оператор перехода JMP на реальную точку входа.
Образ памяти программы типа .COM показан на рис.4.



L----------<- SP=FFFEh

Рис.4. Образ памяти программы .COM.

После загрузки программы все сегментные регистры указывают на начало единственного сегмента, т.е. фактически на начало PSP. Указатель стека автоматически инициализируется числом FFFEh. Таким образом, независимо от фактического размера программы ей выделяется 64 Кбайт адресного пространства, всю нижнюю часть которого занимает стек. Поскольку верхняя граница стека не определена и зависит от интенсивности и способа использования стека программой, следует опасаться затиранием стеком нижней части программы. Впрочем, такая опасность существует и в программах типа .EXE .

4. Ассемблирование и компановка программы.

Правила, приведенные в п.3 с 1) no 6) - это часть синтаксиса ассемблера.
Все действия над программай, начиная с ее ввода и кончая выполнением, описаны ниже и осуществляются с помощью операционной системы MS-DOS.

1) Вводят с клавиатуры программу пользователя, используя программу, называемую редактором текста. Полученный файл называют файлом исходной программы.

2) Транслируют исходную программу (файл), используя программу ассемблер TASM.EXE (MASM.EXE), одновременно с получением распечатки программы на языке ассемблер, помещают в файл полученную программу на машинном языке. Полученный таким образом файл представляет собой файл объектной программы.

3) Связывают по необходимости файл объектной программы с библиотечными программами и другими программами с помощью редактора связей TLINK.EXE (LINK.EXE) и таким образом формируют файл загрузочного модуля в такой форме, при которой сразу возможно его выполнение.

4) Выполняют загрузочный модуль.

Пояснение к этапам реализуемых с помощью MS_DOS.
Для этапа 1) обычно используют редактор текста. При выборе редактора следует иметь в виду, что многие текстовые процессоры (например, Microsoft Word) добавляют в выходной файл служебную информацию. Поэтому следует воспользоваться редактором,
выводящим в выходной файл "чистый текст", без каких-либо управляющих символов. К таким редакторам относятся, например, Лексикон, Norton Editor, Multi-Edit и др. Файл с исходным текстом должен иметь расширение .ASM .

Строка вызова ассемблера имеет вид:

TASM [options] исход_файл[,объект. файл][,файл_лист][,файл REF]

где исходный файл - имя файла с исходным модулем;

объект. файл - имя объектного файла, если его указывать, то предполагается аналогичное имя предыдущего файла;

файл лист - имя файла листинга, в котором не только исходный текст, но также в левой части транслированный машинный шестадцатиричном формате. В левой колонке находятся шестнадцатиричные адреса команд и данных, если не указывать имя файла, то он создается.

файл REF - имя файла листинга перекрестных ссылок, который полезен для очень больших программ, где необходимо видеть, какие команды ссылаются на какие поля данных.

options - параметры, их значения преведены в приложении.

Если в результате ассемблирования не обнаружено ошибок, то следующий шаг - компоновка объектного модуля, для этого вводится команда:

TLINK objfiles, exefile, mapfile, libfales

где objfiles - имена файлов объектного модуля, тип OBJ опускается;
txtfile - имя исполняемого файла с расширением EXE, если не указывается, то сохраняется имя объектного файла;
mapfile - MAP-файл, который содержит таблицу имен и размеров сегментов и ошибок, которые обнаружит компоновщик, для удобства вывод организовать на CON;
libfiles - указываются библиотечные файлы, которые представляются в виде объектных модулей, доступных для извлечения компоновщиком для присоединения к объектному файлу.

Используемые параметры приведены в приложении.

Если вышеуказанные операции повторяются многократно, то для ускорения работы компьютера и облегчения своей работы, использовать возможность

Norton Commander выполнить команду в зависимости от расширения имени файла. При нажатии пользователем клавиши enter в момент, когда выделен какой-либо файл, NC может выполнить некоторую команду в зависимости от расширения имени этого файла. Какая именно командабудет выполнена, указав в файле NC.EXT. Отредактировать самим NC (клавиши F9,C,X) действия вызова TASM и TLINK по расширениям .ASM и .OBJ.

БИБЛИОТЕКАРЬ TLIB

Программа TLIB - это утилита для управления библиотеками и отдельными объектными файлами (библиотекарь). Библиотекарь предоставляет удобный инструмент для работы с набором объектных модулей, как с одним программным модулем.

Библиотеки, включенные в Турбо Ассемблер, построены с помощью утилиты TLIB. Вы можете использовать данную утилиту для построения своих собственных библиотек или их модификации, а также для работы с библиотеками, построенными другими программистами, или приобретенными вами коммерческими библиотеками.

Библиотекарь TLIB позволяет:

- создать библиотеку из отдельных объектных модулей;
- добавить объектные модули или другие библиотеки в существующую библиотеку;
- уничтожить раздел библиотеки;
- заменить раздел библиотеки;
- вывести модуль из состава библиотеки;
- просмотреть оглавление библиотеки.

При изменении содержимого библиотеки TLIB создает копию редактируемой библиотеки с расширением .BAK.

TLIB может также создавать расширенный словарь в библиотеке для увеличения скорости редактирования программ (См. параметр /E).

Хотя утилита TLIB не является существенной для создания выполняемых программ на Ассемблере, ее использование значительно облегчает труд программиста и повышает его производительность. Библиотекарь (и вы в этом убедитесь) незаменим при разработке больших проектов. Если вы работаете с объектными модулями, созданными другими программистами, данная утилита поможет вам при необходимости обслуживать подобные библиотеки.

Библиотекарь можно вызывать, набрав команду TLIB в ответ на подсказку DOS. Для получения полного списка возможностей TLIB, наберите просто TLIB и нажмите клавишу Enter.

Общая форма вызова TLIB следующая:

tlib имя_библ [/C] [/E] [/Pразмер][операции][,файл_листинга]

tlib имя для вызова библиотекаря

имя_библ Имя создаваемой или существующей библиотеки. Это имя должно присутствовать всегда. Специальные символы недопустимы. Если расширение не указано, то по умолчанию оно принимается .LIB. Если библиотека не существует и выполняется операция добавления модуля, то TLIB создает библиотеку.

/C Флаг типа букв. Этот параметр обычно используется.

/E Создание расширенного словаря.

/Рразмер Устанавливает для страниц библиотеки указанный размер.

операции Список операций для TLIB. Операции могут следовать в любом порядке.

файл_листинга Имя файла листинга содержимого оглавления библиотеки. Перед именем файла должна находится запятая. Если вы не укажете имя файла, то листинг выводится не будет. Листинг - это алфавитный список модулей вместе с общими переменными, которые в них объявлены. Расширение принимаемое по умолчанию для этого файла - .LST. Возможно переопределения устройства для вывода листинга при помощи стандартных средств DOS. Листинг можно направить на экран, указав в качестве имени файла листинга CON, или на принтер, задав имя PRN.

Ниже приведены примеры работы с программой TLIB:

1. Создание библиотеки MYLIB.LIB из модулей X.OBJ,Y.OBJ и Z.OBJ:

```
tlib mylib +x +y +z
```

2. Создание библиотеки MYLIB.LIB и получение листинга:

```
tlib mylib +x +y +z ,mylib.lst
```

3. Получение листинга существующей библиотеки CS.LIB:

```
tlib cs,cs.lst
```

4. Замена модуля X.OBJ на новую копию, добавление раздела A.OBJ и удаление раздела Z.OBJ из библиотеки MYLIB.LIB:

```
tlib mylib -+x +a -z
```

ЗАДАНИЕ.

- - - - - - - - -

1). Подготовка простейшей программы типа .EXE и изучение ее структуры. Создайте файл с приведенным ниже текстом программы. Оттранслируйте и скомпонуйте его, получив файл с расширением .EXE. Убедитесь в том, что программа нормально запускается и выполняет предусмотренные в ней действия. Научиться исключать файлы перекрестных ссылок и листинга по переменке или оба сразу в строке вызова транслятора.

```
;Подготовка программы
;TASM имя_ис_прог,,  - ассемблирование исходной программы.
;TLIK имя_об_прог    - компоновка объектной программы.
text    segment 'code'
        assume  cs:text,ds:data,es:data,ss:stack
;Определения
stdout  equ     1               ;Дескриптор стандартного вывода
cr      equ     10              ;Возврат каретки
lf      equ     13              ;Перевод каретки
myproc  proc
        mov     AX,data         ;Инициализируем
        mov     DS,AX           ;сегментный регистр DS
;Выведем на экран строку текста
        mov     AH,40h          ;Функция вывода
        mov     BX,stdout       ;Дескриптор
        mov     CX,meslen       ;Длина сообщения
        mov     DX,offset mes   ;Адрес сообщения
        int     21h             ;Прерывание MS-DOS
;Завершим программу
outprog: mov    AH,4Ch          ;Функция завершения
        mov     AL,00h          ;Код завершения (0)
        int     21h             ;Прерывание MS-DOS
myproc  endp
text    ends
data    segment
;Поля данных
mes     db      'Программа .EXE стартовала',cr,lf
meslen  equ     $-mes
data    ends
stack   segment para stack 'stack'
        db      128 dup (?)     ;Область стека
stack   ends
        end     myproc
```

2). Подготовка простейшей программы типа .COM и изучение ее структуры. Задание аналогично 1).

```
;Подготовка программы
;TASM имя_ис_прог,,
;TLIK /t имя_об_прог
text    segment 'code'
        org     100h
        assume  cs:text,ds:text,es:text,ss:text
;Определения
stdout  equ     1               ;Дескриптор стандартного вывода
cr      equ     10              ;Возврат каретки
lf      equ     13              ;Перевод каретки
myproc  proc
;Выведем на экран строку текста
        mov     AH,40h          ;Функция вывода
        mov     BX,stdout       ;Дескриптор
        mov     CX,meslen       ;Длина сообщения
        mov     DX,offset mes   ;Адрес сообщения
        int     21h             ;Прерывание MS-DOS
;Завершим программу
outprog: mov    AX,4C00h        ;Функция завершения, код
                                ;завершения = 0
        int     21h             ;Прерывание MS-DOS
myproc  endp
;Поля данных
mes     db      'Программа .COM стартовала',cr,lf
meslen  equ     $-mes
text    ends
        end     myproc
```

П Р И Л О Ж Е Н И Е

```
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland International
Syntax:  TASM [options] source [,object] [,listing] [,xref]
/a,/s         Alphabetic or Source-code segment ordering
/c            Generate cross-reference in listing
/dSYM[=VAL]   Define symbol SYM = 0, or = value VAL
/e,/r         Emulated or Real floating-point instructions
/h,/?         Display this help screen
/iPATH        Search PATH for include files
/jCMD         Jam in an assembler directive CMD (eg. /jIDEAL)
/kh#          Hash table capacity # symbols
/l,/la        Generate listing: l=normal listing, la=expanded listing
/ml,/mx,/mu   Case sensitivity on symbols: ml=all, mx=globals, mu=none
```

/mv# Set maximum valid length for symbols
/m# Allow # multiple passes to resolve forward references
/n Suppress symbol tables in listing
/os,/o,/op,/oiObject code: standard, standard w/overlays, Phar Lap, or IBM
/p Check for code segment overrides in protected mode
/q Suppress OBJ records not needed for linking
/t Suppress messages if successful assembly
/uxxxx Set version emulation, version xxxx
/w0,/w1,/w2 Set warning level: w0=none, w1=w2=warnings on
/w-xxx,/w+xxx Disable (-) or enable (+) warning xxx
/x Include false conditionals in listing
/z Display source line with error message
/zi,/zd,/zn Debug info: zi=full, zd=line numbers only, zn=none

TASM [варианты] исх_файл [,объект_файл] [,листинг] [,пер_ссылки]

/a,/s Упорядочивание сегментов по алфавитному порядку или порядку исходного кода
/c Генерация в листинге перекрестных ссылок
/dSYM[=VAL] Определяется SYM = 0 или SYM = VAL
/e,/r Эмулируемые или действительные инструкции с плавающей точкой
/h,/? Выводится данная справочная информация
/lPATH Включаемые файлы ищутся по маршруту, определяемому PATH
/jCMD Определяет начальную директиву Ассемблера (например, jIDEAL)
/kh# Мощность хеш-таблицы #
/l,/la Генерация листинга: l=обычный листинг, la=расширенный
/ml,/mx,/mu Различимость в регистре букв идентификаторов: ml=все, mx=глобальные, mu=не различаются
/mv# Задает максимальную длину идентификаторов
/m# Разрешает выполнение нескольких проходов для удовлетворения опережающих ссылок
/n Подавление в листингах таблицы символов (идентификаторов)
/p Проверка перекрытия сегмента кода в защищенном режиме
/q Подавление записей .OBJ, не требующиеся при компоновке
/t Подавление сообщений при успешном ассемблировании
/w0,/w1,/w2 Задание уровня предупреждение: w0=нет предупреждений, w1=w2=есть предупреждения
/w-xxx,/w+xxx Запрещение или разрешение предупреждения типа xxx

/x Включение в листинги блоков условного ассемблирования

/z Вывод на дисплей строк программы, в которых обнаруживаются ошибки.

/zd Помещение в объектный файл информации о номерах строк (используется при отладке программы с помощью отладчика Turbo Debugger)

/zi Помещение в объектный файл полной информации для последующей отладки с использованием всех возможностей Turbo Debugger

/zn Запрет помещения отладочной информации в объектный файл.

С помощью параметров командной строки вы можете задавать имя одного или нескольких ассемблируемых файлов, а также параметры, управляющие их ассемблированием.

Общий вид командной строки выглядит следующим образом:

TASM файлы [; файлы]...

Точка с запятой после левой квадратной скобки позволяет вам в одной командной строке ассемблировать несколько групп файлов. По желанию вы можете задать для каждой группы файлов различные параметры, например:

TASM /E FILE1; /A FILE2

В общем случае группа файлов в командной строке может иметь вид:

[параметр]...исх_файл [[+] исходный_файл]...
[,[объектный_файл] [, [файл_листинга],
[, [файл_перекрестных_ссылок]]

Этот синтаксис показывает, что группа файлов может начинаться с любого параметра, который вы хотите применить к этим файлам, а затем могут следовать файлы, которые вы хотите ассемблировать. Именем файла может быть одно имя файла, либо вы можете использовать обычные трафаретные символы DOS * и ? для задания группы ассемблируемых файлов. Если расширение имени файла не указано, Турбо Ассемблер использует по умолчанию расширение .ASM.

TASM MYFILE,,,MYXREF

По этой команде файл MYFILE.ASM ассемблируется в файл MYFILE.OBJ, листинг выводится в файл с именем MYFILE.LST, а перекрестные ссылки - в файл MYXREF.XRF.
Если при спецификации ассемблируемых исходных файлов вы используете трафаретные символы, их можно использовать также для задания имен файла листинга и объектного файла. Например, если в текущем каталоге содержатся файлы XX1.ASM и XX2.ASM, то командная строка:

TASM XX*,YY*

ассемблирует все файлы, начинающиеся с букв XX, генерирует объектные файлы, имена которых будут начинаться с YY, а остальную часть имени формирует в соответствии с именем исходного файла. Результирующие объектные файлы получат, таким образом, имена YY1, OBJ и YY2.OBJ.
Если вы не хотите создавать объектный файл, но хотите получить файл листинга, или если вы хотите получить файл перекрестных ссылок, но не хотите создавать файл листинга или объектный файл, можно в качестве имени файла задать нулевое (фиктивное) устройство. Например:

TASM FILE1,,NUL,

Эта команда ассемблирует файл FILE1.ASM в объектный файл FILE1.OBJ. При этом файл листинга не создается, а создается файл перекрестных ссылок FILE1.XRF.

Параметры командной строки

Необязательные параметры командной строки позволяют вам управлять поведением Ассемблера, а также тем, какую информацию он выводит на экран, в листинг и объектный файл. В Турбо Ассемблере предусмотрены некоторые параметры, которые не выполняют никаких действий, а используются только для совместимости текущей версии TASM с предыдущими версиями MASM (макроассемблер фирмы Microsoft):

/B Задает размер буфера
/V Выводит на экран дополнительную статистику

Вы можете задавать параметры, представляющие собой любую комбинацию букв в верхнем и нижнем регистре. Кроме того, параметры можно задавать в любом порядке (кроме параметров /I и /J), они будут при этом обрабатываться последовательно. При использовании параметра /D нужно быть внимательным: идентификаторы надо определить до того, как они будут использованы в последующих параметрах /D.
Примечание: С помощью директив, указанных в вашем исходном коде, вы можете отменить эквивалентные им параметры Ассемблера.
Утилита TLINK является отдельной программой и может быть использована как автономный компоновщик (редактор связей). По умолчанию TLINK вызывается после успешной компиляции программы для объединения объектных модулей и генерации выполняемого файла.

В этом приложении описывается использование утилиты TLINK, как автономного компоновщика.

При запуске утилиты TLINK без параметров, она выводит на экран все параметры. Например:

```
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Syntax:  TLINK objfiles, exefile, mapfile, libfiles
@xxxx indicates use response file xxxx
Options: /m - map file with publics                  (1)
        /x = no map file at all                    (2)
        /i = initialize all segments               (3)
        /l = include source line numbers           (4)
        /s = detailed map of segments              (5)
        /n = no default libraries                  (6)
        /d = warn if duplicate symbols in libraries  (7)
        /c = lower case significant in symbols       (8)
        /3 = enable 32-bit processing              (9)
        /v = include full symbolic debug information  (10)
        /e = ignore Extended Dictionary            (11)
        /t = create COM file                       (12)
        /o = overlay switch
        /ye = expanded memory swapping
        /yx = extended memory swapping
```

1 - файл MAP с общедоступными идентификаторами; 2 - нет файла MAP; 3 - инициализировать все сегменты; 4 - включать номера исходных строк; 5 - подробная схема сегментов; 6 - нет используемых по умолчанию библиотек; 7 - предупреждение при обнаружении дублирования идентификатора в библиотеке; 8 - различие в идентификаторах строчных и прописных букв; 9 - разрешение 32-разрядной обработки; 10 - включить полную информацию для отладки; 11 - игнорировать расширенный словарь; 12 - создать файл COM.

Формат командной строки следующий:

TLINK объектные_файлы, выполн_файл, MAP_файл, библиотеки

Например, вы ввели следующую строку:

```
tlink /c mainline wd ln tx,fin,mfin,lib\comm lib\support
```

TLINK будет интерпретировать эту строку следующим образом:

- необходимо учитывать "размер" букв (/c);

- редактируются объектные файлы: MAINLINE.OBJ, WD.OBJ,LN.OBJ и TX.OBJ;

- выполняемый файл будет назван FIN.EXE;

- файл карты загрузки (схемы программы) будет назван MFIN.MAP;

- во время редактирования будут использованы библиотеки COMM.LIB и SUPPORT.LIB. Они находятся в каталоге LIB.

Компоновщик TLINK предполагает по умолчанию следующие расширения для файлов:

- .OBJ для объектных файлов;
- .EXE для исполняемых файлов;
- .MAP для файлов схемы программы;
- .LIB для библиотек.

Если вы не определили имя файла .EXE, то оно будет образовано из имени первого объектного файла, при помощи добавления к нему расширения .EXE.

Когда вы используете параметр /t, выполняемый файл по умолчанию будет иметь расширение .COM, а не .EXE.

ЗАДАЧИ ПО ПРОГРАММИРОВАНИЮ НА ЯЗЫКЕ АССЕМБЛЕРА.

Задача 1. Циклы. Создать текстовый символьный массив (64 символа, символа пробела до символа _). Вывести созданную строку на экран.

```
;Основные фрагменты программы
;Заполним строку
    mov   CX,64       ;Число символов
    mov   AL,' '      ;Первый символ - пробел
    mov   SI,0        ;Начальное смещение в строке
pov: mov   mes[SI],AL  ;Перешлем байт в строку
    inc   SI          ;Увеличиваем на 1 - обращение
                      ;к следующему байту
    inc   AL          ;Следующий символ
    loop  pov         ;Цикл CX раз
;Выведем строку mes на экран
    mov   AH,40h      ;Файловая функция вывода
    mov   BX,1        ;Дескриптор стандартного вывода
```

```
    mov   CX,64       ;Столько байт вывести
    mov   DX,offset mes ;Адрес строки
    int   21h         ;Выход в MS-DOS
;Завершим программу
    . . .
;Поля данных
mes   db    64 dup('~')  ;Строка с контрольным содержимым
```

Задача 2. Циклы. Просмотреть изображение символов второй половины кодовой таблицы (коды 128 - 255, всего 128 символов). Для этого создать текстовый символьный массив, состоящий из кодов этих символов, и вывести его на экран.

Задача3. Вложенные циклы. Создать программную задержку. Определить значение параметров программы, позволяющие получить задержки 10, 3 и 1 с.

```
;Основные фрагменты программы
;Организуется демонстрационный цикл из 10 шагов
      mov     CX,10
loop1:  push    CX
;Вывод строки nes на экран
      . . .
;Организуем программную задержку
      mov     CX,time       ;Число шагов внешнего цикла
outer:  push    CX          ;Сохраним его в стеке
      mov     CX,0          ;64 Кбайт во внутреннем цикле
inner:  loop    inner       ;Тело цикла - из одной строки
      pop     CX            ;Восстановим CX перед ком.
      loop    outer         ;loop внешнего цикла outer
      pop     CX            ;Восстановим CX демонстраци-
      loop    loop1         ;онного цикла loop1
;Поля данных
mes     db      '<>'        ;Демонстрационная строка
time    dw      10          ;Настройка времени задержки
```

Задача 4. Пересылка строк. Переслать строку(массив) произвольной длины, включив ее в состав другой, более длинной строки. Вывести полученную строку на экран.

```
;Основные фрагменты программы
;Перешлем строку
      mov     CX,strlen     ;CX=длина строки
      mov     AX,DX         ;Настроим сегментный ре-
      mov     ES,AX         ;гистр ЕС на наши данные
```

```
        mov     SI,offset str  ;DS:SI=адрес исходной строки
        lea     DI,mes+3       ;ES:DI=адрес пересылки
rep:    movsb                  ;Пересылка CX байт
;Выведем строку на экран
        mov     AH,40h         ;Файловая функция вывода
        mov     BX,1           ;Дескриптор стандартного вывода
        mov     CX,80          ;Столько байт вывести
        mov     DX,offset mes  ;Адрес строки
        int     21h            ;Переход в MS-DOS
;Поля данных
str     db      ' Пересылаемая строка  '
strlen  equ     $-str          ;Длина строки
mes     db      80 dup ('*')   ;Строка приемник
```

Задача 5. Сравнение строк. Сравнить две одинаковые строки, вывести на экран результат сравнения. Модифицировать поля данных программы, сделав строки не равными и выполнить повторно.

```
;Основные фрагменты программы
;Сравним строки
     mov   CX,32              ;Длина строк
     mov   AX,seg str2        ;Сегментный адрес str2
     mov   ES,AX              ;Настроим на него ES
     mov   SI,offset str1     ;DS:SI=адрес str1
     mov   DI,offset str2     ;ES:DI=адрес str2
repe  cmpsb                   ;Сравнение CX байт
     jne   notequ             ;Если не равны, на notequ
;Выведем на экран сообщение mes1 о равенстве строк
     . . .
     jmp   outprog            ;На завершение программы
notequ:
;Выведем на экран сообщение mes2 о неравенстве строк
     . . .
;Поля данных
str1  db    32 dup ('&')       ;Первая строка
str2  db    32 dup ('&')       ;Вторая строка
mes1  db    'Строки одинаковы',10,13
mes2  db    'Строки различаются!',10,13
```

Задача 6. Составить макроопределения типовых действий (например, программной задержки и вывода строки на экран). Написать программу, содержащую макровызовы с разными значениями параметров и проверить правильность ее выполнения.

```
;Основные фрагменты программы
```

```
;Макроопределение для программной задержки
;Параметр=задержка в секундах
delay macro time              ;Имя delay, параметр time
      local   outer,inner     ;Локальные метки
      mov     CX,time*3       ;CX=время в сек (коэффициент
                        ;подбирается экспериментально)
outer:  push    CX            ;Сохраним его в стеке
      sub     CX,CX           ;64К шагов во внутреннем цикле
inner:  loop    inner         ;Тело цикла из одной строки
      pop     CX              ;Восстановление CX перед ком.
      loop    outer           ;loop внешнего цикла outer
      endm                    ;Конец макроопределения

;Макроопределение вывода строки на экран
;Первый параметр = адрес строки, второй параметр = ее длина
outstring macro  mes,len
      mov     AH,40h
      mov     BX,1
      mov     CX,len
      mov     DX,offset mes
      int     21h
      endm

;Начало главной процедуры
myproc  proc
      mov     AX,data     ;Инициализация сегментного
      mov     DS,AX       ;регистра DS
      outstring m1,3      ;Вывести строку m1, 3 байта
      delay   10          ;Задержка на 10 с
      outstring m2,4
      delay   5
      outstring m3,1
;Поля данных
m1      db      '<> '
m2      db      '*** '
m3      db      '!'
```

Задача 7. Оформить макроопределения из предыдущей задачи в виде отдельного файла с именем, например, MACRO.LIB, создать тем самым макробиблиотеку пользователя. Написать программу, содержащую обращение к макроопределениям из макробиблиотеки.

```
;Основные фрагменты программы
      include macro.lib
```

```
        . . .
        outstring m1,3
        delay   10
        outstring m2,4
        delay   5
        outstring m3,1
;Поля данных
m1      db      '<> '
m2      db      '*** '
m3      db      '!'
```

Задача 8. Подпрограммы. Оформить фрагменты, организующие программную задержку и вывод строки на экран, в виде подпрограмм, включенных в текст основной программы (в виде отдельных процедур). Вызвать процедуры из главной программы, передавая параметры через выделенные для этого регистры.

```
;Основные фрагменнты программы
;Подпрограмма Delay задержки
;На входе CX=значение задержки в секундах
delay   proc    near
        push    DX      ;Сохраним регистры, неявно исполь-
        push    AX      ;зуемые подпрограммой
        xchg    AX,CX   ;AX=значение задержки
        mov     CX,3    ;Коэффициент перевода в секунды
                        ;(следует подобрать экспериментально)
        mul     CX      ;Умножим CX на AX
        mov     CX,AX   ;Перешлем результат в CX
        pop     AX      ;Восстановим значение AX и
        pop     DX      ;DX из вызывающей программы
outer:  push    CX      ;Далее как и раньше
        mov     CX,0
inner:  loop    inner
        pop     CX
        loop    outer
        ret             ;Возврат в вызывающую процедуру
delay   endp            ;Конец процедуры
;Подпрограмма outstring вывода строки на экран
;На входе CX=число выводимых байт, DX=адрес строки
outstring  proc near
        mov     AH,40
        mov     BX,1
        int     21h
        ret
outstring  endp
```

```
;Начало главной процедуры
myproc  proc
        . . .
        mov     CX,meslen     ;Настроим CX
        mov     DX,offset mes ;Настроим DX
        call    outstring     ;Вызов процедуры вывода строки
        mov     CX,10         ;Настроим CX
        call    delay         ;Вызов процедуры задержки
        mov     CX,meslen-6   ;Еще раз выведем строку
        mov     DX,offset mes+3 ;(измененную)
        call    outstring
;Поля данных
mes     db      '*** Сообщение ***',10,13
meslen  equ     $-mes
```

Задача 9. Оформить подпрограммы из предыдущего примера в виде отдельных исходных файлов. Оттранслировать их и получить объектные модули. С помощью программы-библиотекаря TLIB.EXE создать объектную библиотеку пользователя и включить в нее объектные модули процедур-подпрограмм. Написать главную программу, содержащую вызовы процедур из объектной библиотеки. Скомпоновать главную программу с модулями из объектной библиотеки и проверить ее работоспособность.

Задачи по программированию операций над файлами, каталогами и дисками.

Задача 1. Создание файла. В директории ASM диска создать файл с MYFILE.001 и записать в него символьную строку. После выполнения программы убедится в его наличии и содержимом.

```
;Определения
cr      equ     0Dh
lf      equ     0Ah
text    segment 'code'
        assume  CS:text,DS:data
myproc  proc
        mov     AX,data
        mov     DS,AX
;Создание файла
        mov     AH,3Ch              ;ункция создания файла
        mov     CX,0                ;Без атрибутов
        mov     DX,offset filename  ;Адрес имени файла
        int     21h
        mov     handle,AX           ;Сохраним дескриптор файла
;Запись строки в файл
```

```
        mov     AH,40h              ;Функция закрытия
        mov     BX,handle           ;Дескриптор
        mov     CX,stringln         ;Длина строки
        mov     DX,offset string    ;Адрес строки
        int     21h
;Закроем файл (больше его не используем)
        mov     AH,3Eh              ;Функция закрытия
        mov     BX,handle           ;Дескриптор
        int     21h
;Завершим программу
outprog: mov    AX,4C00h        ;Функция завершения, код за-
                        ;вершения = 0
        int     21h
myproc  endp
text    ends
data    segment
string  db      'Текстовая строка',cr,lf ;Строка для записи в
                                ;файл
stringln equ    $-string            ;Ее длина
handle  dw      ?                   ;Ячейка для дескриптора
filname db      'MYFILE.001',0    ;Имя файла в формате ASCIIZ
data    ends
stack   segment para stack 'STACK'
        db      128 dup (?)
stack   ends
        end     myproc
```

Задача 2. Чтение файла. Прочитать содержимое файла MYFILE. 001 в память и вывести его на экран. Предполагается, что размер файла не более 80 байт.

```
;Определение
stdout  equ     1               ;Дескриптор стандарт. вывода
;Основные фрагменты программы
;Открываем файл
        mov     AH,3Dh          ;Функция открытия файла
        mov     AL,2            ;Доступ для чтения/записи
        mov     DX,offset string ;Адрес имени файла
        int     21h
        mov     handle,AX       ;Получение дескриптора
;Попытка чтения 80 байт
        mov     AH,3Fh          ;Функция чтения
        mov     BX,handle       ;Дескриптор
        mov     CX,80           ;Столько читать
        mov     DX,offset bufin  ;сюда
```

```
        int     21h
        mov     CX,AX           ;Столько реально прочитали
;Вывод прочитанного на экран
        mov     AH,40h          ;Функция записи
        mov     BX,stdout       ;Дескриптор
        mov     DX,offset bufin  ;Отсюда выводить CX байт
        int     21h
;Завершение программы
        ...
;Поля данных
bufin   db      80 dup (' ')    ;Буфер ввода
handle  dw      ?               ;Ячейка для дескриптора
filename db     'MYFILE.001',0   ;Имя файла в формате ASCIIZ
```

Задача 3. Прямой доступ к файлу. Прочитать 8 байт из созданного ранее файла MYFILE.001, НАЧИНАЯ с байта 5, вывести их на экран.

```
;Определение
stdout  equ     1
;Основные фрагменты программы
;Открыть файл
        ...
;Установить указатель
        mov     AH,42h
        mov     AL,0
        mov     BX,handle
        mov     CX,0
        mov     DX,5
        int     21h
;Прчитать 8 байт данных с помощью функции 3Fh
        ...
;Выведем прочитанное на экран с помощью функции 40h
        ...
;Завершить программу
        ...
;Поля данных
bufin   db      80 dup ('*')    ;Буфер ввода
handle  dw      ?               ;Ячейка для дескриптора
filename db     'MYFILE.001',0   ;Имя файла в формате ASCIIZ
```

Задача 4. Добавление данных к файлу. Добавить символную строку к концу символьного файла. Проконтролировать содержимое и длину файла до и после добавления (средствами NC). Повторить выполнение программы и еще раз проконтролировать результат.

```
;Основные фрагменты программы
;Откроем файл с указанным именем функцией 3Dh и сохраним полу-
;ченый дескриптор в ячейке handle
      ...
;Установим указатель на конец файла
      mov    AH,42h        ;Функция установки указателя
      mov    AL,02         ;От конца файла
      mov    BX,handle     ;Дескриптор
      mov    CX,0          ;Старшая половина указателя
      mov    DX,0          ;Младшая половина указателя
      int    21h
;Допишем новую строку
      mov    AH,40h        ;Функция записи
      mov    BX,handle     ;Дескриптор
      mov    CX,stringln   ;Длина строки
      mov    DX,offset string ;Ее адрес
      int    21h
;Завершим программу
      ...
;Поля данных
string  db     'Новая строка'   ;Строка для вывода в файл
stringln equ   $-string        ;Длина строки
handle  dw     ?               ;Ячейка для дескриптора
filename db    'MYFILE.001',0   ;Имя файла
```

Задача 5. Изменение атрибутов файла. Установить у файла MYFILE.001 атрибут "только для чтения". После завершения программы проконтролировать результат с помощью средств NC (DOS).

```
;Основные фрагменты программы
;Установим атрибут "только для чтения"
      mov    AH, 43h       ;Функция работы с атрибутами
      mov    AL,1          ;Установка атрибутов
      mov    CX, 1         ;"Только для чтения"
      mov    DX,offset filename ;Адрес имени файла
      int    2lh
```

Задача 6. Изменение характеристик файла. Изменить дату и время создания файла MYFILE.001.

```
;Основные фрагменты программы
;Откроем файл и сохраним его дескриптор
         ...
;Изменим дату и время создания файла
      mov    AH, 57h    ;Функция даты/времени
```

```
        mov     AL,1        ;Установить дату/время
        mov     BX, handle ;Дескриптор файла
        mov     CX,0        ;Очистим CX
        or      CX,sec      ;Добавим секунды
        or      CX,min      ;Добавим минуты
        or      CX,hour     ;Добавим часы
        xor     DX,DX       ;Очистим DX
        or      DX,day      ;Добавим день
        or      DX,mon      ;Добавим месяц
        or      DX, year    ;Добавим год
        int     21h
;Завершим программу
;Поля данных
fname   db      'MYFILE.001',0 ;Имя файла
handle  dw      ?           ;Ячейка для дескриптора
sec     dw      6/2         ;6 секунд
min     dw      15*32       ;15 минут
hour    dw      16*2048     ;16 часов
day     dw      25          ;25 число
mon     dw      3*32        ;Март
year    dw      13*512      ;13 лет от 1980, т.е. 1993 г.
```

Задача 7. Переименование файла. Переименовать файл MYFILE. 001, находящийся в текущем каталоге, дав ему имя NEWNAME. DAT.

```
;Основные фрагменты программы
;Настроим сегментный регистр ES на наш сегмент данных
        push    DS
        pop     ES
;Переименуем файл
        mov     AH,56h          ;Функция переименования
        mov     DX,offset oldname ;Адрес старого имени
        mov     DI,offset newname ;Адрес нового имени
        int     21h
;Завершить программу
        ...
;Поля данных
oldname db      'MYFALE.001',0    ;Старая спецификация
newname db      'NEWNAME.DAT',0   ;Новая спецификация
```

Задача 8. Пересылка файла в другой каталог на том же диске Переслать файл NEWNAME.DAT из текущего каталога в нижележащий каталог NEWDIR, изменив при этом имя файла на NEWNAME.LEX. Отладив программу, проверить ее возможности, помещая файл в разные каталоги и пересылая его в другие.

```
;Основные фрагменты программы
;Настроим сегментный регистр ES на наш сегмент данных
        ...
;Переименуем файл
        mov     AH,56h          ;Функция переименования
        mov     DX,offset oldname ;Адрес старого имени
        mov     DI,offset newname ;Адрес нового имени
        int     21h
;Завершить программу
        ...
;Поля данных
oldname db      'NEWNAME.DAT',0   ;Старая спецификация
newname db      'NEWDIR\'NEWNAME.LEX',0   ;Новая спецификация
```

Задача 9. Создание и удаление каталогов. Создать в текущем каталоге подкаталог NEWDIR. Удалить из текущего каталога подкаталог OLDDIR.

```
;Основные фрагменты программы
;Создадим новый каталог
        mov     AH,39h          ;Функция создания каталога
        mov     DX,offset newname ;Адрес нового имени
        int     21h
;Удалим ненужный каталог
        mov     AH,3Ah          ;Функция удаления каталога
        mov     DX,offset oldname ;Адрес имени каталога
        int     21h
;Завершить программу
        ...
;Поля данных
oldname db      'OLDDIR'    ;Старая спецификация
newname db      'NEWDIR'    ;Новая спецификация
```

Задача 10. Смена диска. Сделать текущим диск A: .

```
;Сделаем текущим диск A:
        mov     AH,0Eh      ;Функция смены диска
        mov     DL,0        ;Код диска A (A=0,B=1,C=2 и т.д.)
        int     21h
;Завершить программу
        ...
```

Задача 10. Создание метки тома. Создать в корневом каталоге дискеты запись с меткой тома PROGRAMDISK. После выполнения программы проверить наличие метки командами DOS: DIR или VOL.

```
;Основные фрагменты программы
;Создадим "файл" - метку тома
      mov    AH,3Ch      ;Функция создания файла
      mov    CX,8        ;Атрибут метки тома
      lea    DX,labela   ;Адрес имени метки
      int    21h
;Завершим программу
;Поля данных
labela  db     'A:\PROGRAMD.ISK',0
```

Задача 11. Проверка метки тома. Найти в корневом каталоге дискеты запись с меткой тома и убедиться, что на дисковод установлена требуемая дискета.

```
;Основные фрагменты программы
;Установим нашу область передачи диска (DTA), чтобы не
;искать ее в префиксе программного сегмента
      mov    AH,1Ah       ;Функция установки DTA
      mov    DX,offset dta ;Адрес нашей DTA
      int    21h
;НАЙДЕМ МЕТКУ
      mov    AH,4Eh       ;Функция поиска первого файла
      mov    CX,8         ;Атрибут "метки тома"
      mov    DX,offset dname ;Адрес спецификации файла
      int    21h
      jc     nolabel            ;Метка отсутствует
;Сравним метки
      mov    SI,offset lbl   ;Адрес имени требуемой метки
      mov    AX,seg_dta      ;Настройка сегментного регист-
      mov    ES,AX           ;ра ES на наш сегмент
      mov    DI,offset dta+1Eh ;Место для имени файла в DTA
      mov    CX,lbllen       ;Длина имени метки
      cld                ;Будем сравнивать в перед
repe  cmpsb              ;Сравнение
      jne    error         ;Метки не совпадают
;Вывести сообщение mes1 о том, что на дисководе стоит требуе-
;мая дискета
      ...
      jmp    outprog
nolabel:
;Вывести сообщение mes2 об отсутствии метки на дискете
      ...
      jmp    outprog
error:
;Вывести сообщение о том, что на дисководе не та дискета
```

```
        ...
;Завершить программу
        ...
;Поля данных
dname   db      'A:\*.*',0 ;
dta     db      43 dup(0)
lbl     db      'PROGRAMD.ISK'
lbllen  equ     $-lbl
mes1    db      'На дисководе стоит правильная дискета',cr,lf
mes1len equ     $-mes1
mes2    db      'На дискете НЕТ метки!',cr,lf
mes2len equ     $-mes2
mes3    db      'На дисководе НЕ ТА дискета!',cr,lf
mes3len equ     $-mes3
```

Задачи по программированию ввода с клавиатуры

Задача 5.1.
Ввод с клавиатуры и вывод на экран символьной информации.
Вывести на экран через дескриптор стандартной ошибки служебное сообщение.Ввести с клавиатуры через дескриптор стандартного ввода символьную строку, вывести её на экран через дескриптор стандартного вывода. Изучить действие операторов перенаправления ввода и вывода, выполнив следующие действия:
- перенаправить вывод программы во вновь создаваемый файл;
- перенаправить вывод программы в имеющийся файл с добавлением новой строки к содержимому файла;
- перенаправить ввод программы, получая данные для неё не с клавиатуры, а из файла;
- перенаправить и ввод и вывод программы, получая данные из одного файла и занося их в другой (в сущности, реализация копирования символьных файлов).

```
;определения
stdin   equ     0               ;дескриптор стандартного ввода
stdout  equ     1               ;дескриптор стандартного вывода
stderr  equ     2               ;дескриптор стандартной ошибки
;Выведем служебное сообщение msg
        ...
;Поставим запрос на ввод строки в буфер buf
        mov     AH,3Fh          ;Функция ввода
        mov     BX,stdin        ;Дескриптор стандартного ввода
        mov     CX,80           ;Ввод максимум 80 байт
        mov     DX,offset buf   ;Адрес буфера ввода
        int     21h
```

```
        mov   actlen,AX    ;Фактически введено
;Выведем на экран
        mov   AH,40h               ;Функция вывода
        mov   BX,stdout    ;Дескриптор стандартного вывода
        mov   CX,actlen    ;Длинна сообщения
        mov   DX,offset buf        ;Адрес сообщения
        int   21h
;Завершим программу
        ...
;Поля данных
msg   db    'Вводите! '
msglen        equ   $-msg          ;Длинна сообщения
buf   db    80 dup (0)   ;Буфер ввода
actlen dw   0
```

Задача 5.2. Ввод с клавиатуры, обработка и вывод на экран символьной информации.

Усложнить программу из примера 5.1, предусмотрев фильтрацию входного символьного потока, например, преобразование строчных латинских (или русских, или и тех, и других) букв в прописные. Отладив программу, и используя её вместе с предидущей,изучить процедуру конвейеризации программ на примере следующих операций (программа из примера 5.1 названа P51, из примера 5.2 - P52):

1) P51|P52
2) P51|P52>FILE1.TXT
3) P51<FILE2.TXT |P52
4) P51<FILE2.TXT |P52>FILE3.TXT

Файл FILE2.TXT, содержащий, в частности, строчные буквы, следует создать либо с помощью той же программы P51, либо с помощью любого техтового редактора.

```
;Определения stdin, stdout, stderr
        ...
;Основные фрагменты программы
;Выведем служебное сообщение msg
        ...
;Поставим запрос на ввод строки в буфер buf, отправив длинну
;фактичес введённой строки в actlen
        ...
;Превратим строчные латинские буквы в прописные
        mov   CX,actlen            ;Длинна введённой строки
        mov   SI,0                 ;Указатель в буфере
filter: mov   AL,buf[SI]           ;Возмём символ
        cmp   AL,'a'               ;Меньше 'a'?
        jb    nolet                ;Да, не преобразовывать
```

```
        cmp   AL,'z'            ;Больше 'z'?
        ja    nolet             ;Да, не преобразовывать
        sub   AL,20h                  ;Преобразуем в прописную
        mov   buf[SI],AL        ;И отправим назад в buf
nolet:  inc   SI                ;Сместим указатель
        loop  filter            ;Цикл
;Выведем введённое на экран
        ...
;Завершим программу
        ...
;Поля данных
msg     db    'Войдите! '
msglen  equ   $-msg                   ;Длинна сообщения
buf     db    80 dup (0)        ;Буфер ввода
actlen  dw    0
```

Задача 5.3. Посимвольный ввод с эхом.
Ввести в программу один символ с клавиатуры. Проанализировать его.Если введено <E>, завершить программу; в случае ввода любого другого символа повторить ввод. После отладки программы проверить её реакцию на ввод <Ctrl>/c. Исследовать работу программы в случае перенаправления её ввода или вывода в файл. Исследовать реакцию программы на ввод с клавиатуры <Ctrl>/c в случае перенаправления её ввода и вывода. Исследовать реакцию программы на ввод с упреждением, для чего сразу после ввода команды активизации программы, но ещё до её фактического запуска нажать на какие-то клавиши.

```
;Основные фрагменты программы
;Введём и проанализируем его на код <E>
;Если введено <E>, завершим программу
again:  mov   AH,01h                  ;Функция ввода с эхо
        int   21h
        cmp   AL,'E'                  ;Введено <E>?
        je    outprog                 ;Да
        jmp   again
outprog:
;Завершим программу
        ...
```

Задача 5.4. Посимвольный ввод без эха.
Заменить в примере 5.3 функцию ввода с эхом 01h на функцию фильтрованного ввода без эха 08h. После отладки проверить реакцию программы на ввод <Ctrl>/c. Для удобства работы предусмотреть в программе вывод на экран запроса на ввод символа.

```
;Основные фрагменты программы
;Выведем запрос req на ввод символа
        ...
;Выведем символ и проанализируем его на код <E>
;Если введено <E>, завершим программу
again: mov   AH,08h
       int   21h
       cmp   AL,'E'
       je    outprog
       jmp   again
outprog:
;Завершим программу
       ...
;Поля данных
req    db    'Введите команду: '
reqlenequ    $-req
```

Задача 5.5. Управление программой от функциональных клавиш.
С помощью функции 08h организовать ввод в программу управляющих кодов от функциональных клавиш <F1>...<F10> или других клавиш, дающих расширенные коды ASCII (сочетания <Alt>/<буква>, <Alt>/<цифра> и др.) Вводимые коды использовать для управления ходом программы.

```
;Основные фрагменты программы
;Ожидаем нажатия клавиши
again: mov   AH,08h                ;Функиция ввода без эха
       int   21h
       cmp   AL,0                  ;Младший байт кода = 0?
       jne   again                 ;Нет, повторить
       mov   AH,08h                ;Да, введём старший байт кода
       int   21h
       cmp   AL,59                 ;Нажата <F1>?
       je    f1                    ;Да
       cmp   AL,84                 ;Нажаты <Shift>/<F1>?
       je    shiftf1               ;Да
       cmp   AL,30                 ;Нажаты <Alt>/<A>?
       je    alta                  ;Да
       cmp   AL,120                ;Нажаты <Alt>/<1>?
       je    alt1                  ;Да
       jmp   again                 ;Нажато незапланированное
;Вывод соответствующих сообщений
f1:
;Вывод сообщения mf1
       ...
       jmp   outpr
```

```
shiftf1:
;Вывод сообщения mshf1
        ...
        jmp    outpr
alta:
;Вывод сообщения malta
        ...
        jmp    outpr
alt1:
;Вывод сообщения malt1
        ...
        jmp    outpr
outpr:
;Завершим программу
        ...
;Поля данных
mf1         db     'Введено <F1>'
mf1len              equ    $-mf1
mshf1       db     'Введено <Shift>/<F1>'
mshf1len    equ    $-mshf1
malta       db     'Введено <Alt>/A'
maltalen    equ    $-malta
malt1       db     'Введено <Alt>/1'
malt1len    equ    $-malt1
```

Задача 5.6 Ввод клавиатуры с предварительной очисткой буфера.

Организовать цикл ввода в программу данных по её запросу. Анализировать введённый символ. Если введено <Q>, завершить программу. После отладки проверить реакцию программы на ввод с упреждением. Проверить возможность управления программой кодами, вводимыми через цифровую клавиатуру (при нажатой клавише <Alt>).

```
;Основные фрагменты программы
;Выведем запрос req на ввод символа
        ...
;Очистим буфер ввода и поставим запрос на ввод с клавиатуры
again:  mov    AH,0Ch              ;Функция ввода с отчисткой буфера
        mov    AL,01h              ;Выберем функцию ввода 01h
        int    21h           ;(можно 01h, 06h, 07h, 08h)
;Проанализируем введённое
        cmp    AL,'Q'              ;Введено <Q>?
        je     outprog             ;Да
        jmp    again
outprog:
```

```
;Завершим программу
        ...
;Поля данных
req     db      10,13,'Вводите команду:'
reqlenequ       $-req
```

Задача 5.7. Чтение двухбайтового кода из кольцевого буфера ввода.
Программа, несмотря на примитивность, позволяет наглядно ознакомиться с принципами трансляции scan-кодов программой INT 09h. Для этого программу следует запустить в отладчике CodeView, установить режим индикации регистров процессора и, многократно выполняя строки программы, наблюдать, какие двухбайтовые коды образуются при нажатии различных клавиш и их комбинаций. Любопытно посмотреть коды таких клавиш, как <ESC>, <TAB>, <PgDn>, <Home>, <End>, клавиш со стрелками, а также сочетание управляющих (<Ctrl>, <Alt>, <Shift>) и <обычных> клавиш.

```
;Основные фрагменты программы
;Будем ждать ввода символа
again:mov   AH,00h                  ;Функция чтения двухбайтового кода
        int     16h
        jmp     again
```

Задача 5.8. Управление циклической программой от клавиатуры с помощью функции чтения состояния клавиатуры.
Организовать выход из циклического участка программы при нажатии на любую клавишу. При желании можно усложнить задачу, введя в программу анализ нажатой клавиши и переход на то или иное продолжение программы в зависимости от результатов анализа. В приведённом примере такой анализ отсутствует. Любопытно также удалить из программы строки извлечения символа из кольцевого буфера (функция 00h) и проанализироиать работу такого варианта программы.

```
;Определение
stdoutequ   1               ;Дескриптор стандартного вывода
;Основные фрагменты программы
;Организуем цикл каких-нибудь действий с периодическим опросом
;клавиатуры
;Конкретные действия-вывод символа на экран с небольшой задержкой
again:mov   AH,40h              ;Функция вывода
        mov   BX,stdout     ;Десриптор стандартного вывода
        mov   CX,1          ;Один символ
        mov   DX,offset sym     ;Адрес символа
        int     21h
;Организуем небольшую задержку
```

```
        ...
;Получим состояние клавиатуры
        mov   AH,01h            ;Функция чтения состояния
        int     16h             ;клавиатуры
        jz      again           ;Если Z=1, символа нет
;Сообщим о выходе из цикла с помощью функции 40h
        ...
;Заберём введённый символ из кольцевого буфера ввода
        mov   AH,00h            ;Получим символ в AX
        int     16h             ;и пусть он пропадает
;Завершим программу
        ...
;Поля данных
sym   db      '*'
mes   db      'Программа завершена оператором'
```

Задачи по защите программ от копирования и несанкционированного использования.

Задача 1: защита программ от копирования путем привязки к ее местоположению на жестком диске.

Написать рабочую программу WORK1.COM, выводящую на экран некоторый текст. В начале выполнения программа орпределяет номер своего первого кластера и сравнивает его с числом, записанным в ее полях данных в процессе установки на диске данной копии программы. Если эти числа несовпадают, программа аварийно завершается с выдачей соответствующего сообщения.

Написать установочную программу INSTALL1.EXE, которая орпределяет начальный кластер файла рабочей программы WORK1.COM и записывает его в поле данных рабочей программы.

Вызвать рабочую программу WORK1.COM, убедиться в том, что в неустановленном состоянии она аварийно завершается.

Установить конкретную копию программы WORK1.COM с помощью программы INSTALL1.EXE. Убедиться, что она установленная рабочая программа функционирует нормально.

Скопировать рабочую программу в другой каталог. С помощью команд DOS COMP или FC убедиться в полной тождественности двух копий программ WORK1.COM. Проверить работу "незаконно скопированной" (неустановленной) и исходной, установленной копий программы WORK1.COM. Установить вторую копию рабочей программы и проверить ее работу.

Программа WORK1.COM

```
;Рабочая программа. Перед запуском требует установки
;на жестком диске
```

```
;Основные фрагменты программы
text   segment 'code'
       assume cs:text,ds:text
       org 100h
myproc     proc
          jmp entry         ;Обойдем слово для ключа
cluster    dw   0           ;Слово для ключа
entry:
;откроем файл WORK1.COM с рабочей программой, чтобы он
;попал в таблицу файлов
       mov  AH,3Dh          ;Функция открытия файла
       mov  AL,2            ;Доступ для чтения/записи
       lea  DX,fname        ;Поле с именем файла
       int  21h
       jnc  go1             ;Файл открылся нормально
       jmp  notopen         ;Файл не открылся
;получим доступ к таблице файлов
go1:   mov  AX,5200h        ;Не документированная функция
       int  21h
       mov  DI,ES:[BX+4]    ;Адрес первой таблицы файлов
       mov  AX,ES:[BX+6]    ;Сегмент первой таблицы файлов
       mov  ES,AX           ;ES:DI -> первую таблицу файлов
;обрабатываем таблицы файлов по очереди
;сначала сохраним адрес следующей таблицы
next:  mov  AX,ES:[DI]      ;Смещение к следующей таблице
                            ;или FFFFh для последней
       mov  word ptr next_blk,AX  ;Сохраним его
       mov  AX,ES:[DI+2]    ;Сегмент следующей таблицы
       mov  word ptr next_blk+2,AX ;Сохраним его
;и организуем поиск имени нашего файла в текущей таблице
       mov  CX,ES:[DI+4]    ;Количество блоков управления
                            ;файлами в этой таблице
       add  DI,6            ;ES:DI -> первый блок управления файлами
cmpfile:
       lea  SI,workcom      ;Адрес имени файла
       call compname        ;Сравним имена
       je   found           ;Нашли
       add  DI,59           ;Перейдем к следующему блоку
       loop cmpfile         ;Цикл по блокам этой таблицы
       cmp  word ptr next_blk,0FFFFh ;Конец таблицы
       jne  go2             ;Нет, перейдем к следующей таблице
       jmp  notfnd          ;Таблицы кончились, а нашего файла нет
;настроим ES:DI на следующую таблицу файлов
go2:   mov  DI,word ptr next_blk
       mov  AX,word ptr next_blk+2
```

```
        mov  ES,AX
        jmp  next           ;Будем повторять поиск имени нашего файла
;наконец наш файл найден
found: add  DI,11           ;DI -> номер первого кластера
        mov  AX,cluster     ;Получим ключ
        cmp  AX,ES:[DI]     ;Сравним с истинным номером
        jne  notins         ;Ключи не совпадают
;ключи совпадают, программа установлена и с ней можно работать
;выведем сообщение mes (функция 40h, int 21h)
        ;Далее должна идти содержательная часть рабочей программы.
        ;В данной задаче ее может и не быть
        . . .

;заверши мпрограмму обычным образом
        . . .
notins:
;вывод сообщения mes1 о том, что файл не открылся
        . . .
        jmp  outprog
notopen:
;вывод сообщения mes2 о том, что файл не открылся
        . . .
        jmp  outprog
notfnd:
;вывод сообщения mes3 о том, что файл рабочей программы
;не найден в таблице файлов
        . . .
        jmp  outprog
;Подпрограмма сравнения имен файлов.
;На входе :
;DS:DI -> имя в программе
;ES:DI -> начало блока управления  управления файлами
;на выходе:
;при равенстве имен ZF=1
;при неравенстве ZF=0
compname:
        push  CX        ;Сохраним регистры, которые
        push  DI        ;понадобятся в подпрограмме
        add   DI,32     ;DI -> имя файла
        mov   CX,11     ;Длина имени файла
        cld             ;Сравнение вперед
repe   cmpsb
        pop   DI        ;Восстановим
        pop   CX        ;регистры
        ret             ;Возврат в основную программу
```

```
myproc endp
;Поля данных
next_blk  dd 0
fname     db 'work1.com',0 ;Имя файла в формате файловых функций
workcom   db 'WORK1   COM' ;Имя файла, как оно записано таблице файлов
mes       db 'Программа установлена и будет работать нормально',10,13
meslen    equ $-mes
mes1      db 'Программа не установлена и не может быть запущена',10,13
mes1len   equ $-mes1
mes2      db 'Файл WORK1.COM не открылся',10,13
mes2len   equ $-mes2
mes3      db 'Файл WORK1.COM не найден в таблице файлов',10,13
mes3len   equ $-mes3
text      ends
          ends myproc
```

Программа INSTALL1.EXE

```
;Установочная программа, предназначенная для установки
;на жестком диске рабочей программы WORK1.COM
;Основные фрагменты программы
;Откроем наш файл, чтобы он попал в таблицу файлов
        ...
;Получим доступ к таблице файлов
        ...
;Обрабатываем таблицы файлов по очереди
;Сначала сохраним адрес следующей таблицы
        ...
;И организуем поиск имени нашего файла в текущей таблице
        ...
;Найдем имя нашего файла в таблице файлов
        ...
;Настроим ES:DI на следующую таблицу файлов
        ...
;Файл найден
found:  add     DI,11         ;DI -> номер первого кластера
        mov     AX,ES:[DI]    ;Получим его
        mov     startclst,AX  ;и отправим в startclst
;У нас есть номер первого кластера файла WORK1.COM, но нет
;соответсвующего номера сектора - ни логического, ни физического.
;Прочитаем загрузочную запись нашего диска в буфер sysarea,
;чтобы определить характеристики диска:
;число FAT, размер корневого каталога, число секторов в кластере
        mov     AL,4          ;Дисковод E:
        mov     CX,1          ;Число читаемых секторов
        mov     DX,0          ;Логический номер сектора
```

```
        lea     BX,sysarea    ;Буфер для чтения сектора
        int     25h           ;Абсолютное чтение с диска
        pop     AX            ;Подправим стек после int 25h
;Найдем относительный номер начального сектора нашего файла
        mov     accum,1       ;Пропустим загрузочный сектор
        mov     AL,sysarea+10h ;Число FAT
        xor     AH,AH         ;Будет умножение слов
        mul     word ptr sysarea+16h ;умножим на число секторов в FAT
        add     accum,AX      ;Добавим в аккумулятор
        mov     ax,word ptr sysarea+11h  ;Число записей в корневом
                              ;каталоге
        xor     DX,DX         ;Будет деление слов
        mov     BX,16         ;Число записей на сектор
        div     BX            ;AX=число секторов в корневом каталоге
        add     accum,AX      ;Получили полный размер системной
                              ;области в секторах
        mov     AX,startclst  ;Первый кластер нашего файла
        sub     AX,2          ;Нумерация кластеров с 2
        mov     BL,sysarea+0dh ;Число секторов в кластере
        xor     BH,BH         ;Будет умножение слов
        mul     BX            ;Число секторов от системной
                              ;области до нашего файла
        add     accum,AX      ;Получили относительный номер
                              ;первого сектора файла
;Запишем номер начального кластера из startclst в ячейку
;cluster файла с программой WORK1.COM. Сначала прочитаем
;первый сектор WORK1.COM по его относительному номеру
        mov     AL,4          ;Дисковод E:
        mov     CX,1          ;Число читаемых снкторов
        mov     DX,accum      ;Логический номер сектора
        lea     BX,sysarea    ;Воспользуемся тем же буфером
        int     25h           ;Аьсолютное чтение с диска
        pop     AX            ;Подправим стек после Int 25h
;Запишем в ячейку cluster номер кластера
        mov     AX,startclst
        mov     word ptr sysarea+3,AX;3 байта занимает код команды
                              ;jmp entry в программе WORK1.COM
;Перешлем исправлунный сектор назад на диск
        mov     AL,4          ;Дисковод E:
        mov     CX,1          ;Число записываемых секторов
        mov     DX,accum      ;Логический номер сектора
        lea     BX,sysarea    ;ИЗ того же буфера
        int     26h           ;Аьсолютное запись на диск
        pop     AX            ;Подправим стек после Int 26h
;Выведем сообщение mes о нормальном завершении программы
```

```
        . . .
;Завершим программу обычным образом
        . . .
notopen:
;Вывод сообщения mes2 о том , что файл WORK1.COM не открылся
        . . .
      jmp outprog
notfnd:
;Вывод сообщения mes3 о том , что файл WORK1.COM не найден
;в таблице файлов
        . . .
      jmp outprog
;Подпрограмма сравнения имён файлов
        . . .
;Поля данных
next_blk dd 0
fname    db      'work1.com',0 ;Имя файла в формате файловых функций DOS
workcom  db       'WORK1.COM'   ;Имя файла КАК ОНО ЗАПИСАНО
                        ;в таблице файлов
sysarea  db       512 dup (0)  ;буфер для чтения сектора
accum    dw       0            ;ячейка - аккумулятор
startclst dw      0           ;начальный кластер файла
                        ;WORK1.COM
mes      db      'Программа WORK1.COM установлена на'
       db      'жёстком диске',10,13
meslen   equ     $-mes
mes2     db      'Файл WORK1.COM не открылся',10,13
mes2len  equ     $-mes2
mes3     db      'Файл WORK1.COM не найден в таблице файлов',10,13
mes3len  equ     $-mes3
```

Задача 2.

Защита программ от копирования путём записи ключа в свободное пространство кластера за пределами файла с программой.

Написать рабочую программу WORK2.EXE(.COM),выводящую на экран некоторый текст.В начале выполнения программа считывает последний кластер файла WORK2.EXE(.COM),находит первое слово за логическим концом файла , считывает из него ключ и сравнивает с ключом ,записанным впрограмме.Если ключи не совпадают, программа аварийно завершается с выдачей соответствующего сообщения.

Написать установочную программу INSTALL2.EXE,которая определяет последний кластер файла рабочей программы WORK2.EXE,считывает с диска этот кластер, записывает его за логическим концом файла заданный ключ и переносит модифицированный кластер на диск.

Программа WORK2.EXE

```
;Рабочая прог.Перед запуском требует установки
;на жёстком диске
;Основные фрагменты программы
;Откроем файл WORK2.EXE с рабочей прог.,чтобы
;он попал в таблицу файлов
        . . .
;Получем доступ к таблице файлов
        . . .
;Обрабатываем таблици файлов по очереди
;Сначало сохраним адрес следующей таблици
        . . .
;И организуем поиск имени нашего файла в текущей таблице
        . . .
;Настроим ES:DS на следующий блок
        . . .
;Прочитаем с помощью прерывания Int 25h загрузочную запись
;нашего диска в буфер 'sysarea',чтобы определить размер
;кластера на данном жестком диске
found:
        . . .
mov  AL,sysarea+0Dh  ;  Сохраним размер кластера
mov  clustsize,Al    ;   в ячейке clustsize

;Определим число  подобных  кластеров в файле и число байт
;файла в последнем неполном кластере
;Сначала найдём длинну кластера в байтах

mov  AL,clustsize     ;Число секторов в кластере
xor  AL,AH           ;Будет умножение слов
mov  BX,512           ;В секторе всегда 512 байт
mul  BX              ;Результат умнажения на AX
mov  clustbyte,AX     ;Сохраним его

;Теперь получим двухсловную длинну программы и поделим
;на размер кластера

mov  AX,word ptr ES:[DI+17]    ;Младшая часть длинны
mov  DX,word ptr ES:[DI+17]    ;Старшая часть длинны
div  clustbyte                 ;Разделим на размер кластера
mov  progsize,AX               ;Целое число кластеров в файле
mov  endic,DX                  ;Остаток байт сверх целого
                               ;числа кластеров
inc  progsize                  ;Число кластеров в файле
                               ;включая последний (неполный)
```

```
;Прочитаем последовательно все кластеры файла до последнего
  mov  CX,progsize            ;Число кластеров в файле
read:  push CX                ;Временно сохраним его
  mov  AH,3Fh          ;Обычная файловая функция чтения
  mov  BX,handle              ;Дескриптор открытого файла
                      ;WORK2.EXE
  mov  CX,clustbyte           ;Число байтов в кластере
  lea  DX,buffer             ;Буфер размером 8 секторов
                      ;(Лучше бы выделить динамически
  int  21h                   ;столько,сколько надо
  pop  CX                     ;Восстановим счётчик кластеров
  lop  read                   ;Повторим столько раз,сколько
                      ;кластеров в файле

;Теперь в buffer последний кластер, а в таблице файлов-его номер
;По номеру кластера найдём логический номер
;его последнего сектора
;Сначала определим размер системной области

  mov  accum,1                ;Пропустим загрузочный сектор
  mov  AL,sysarea+10h          ;Число FAT
  xor  AH,AH                  ;Будет умножение слов
  mul  word ptr sysarea+16h     ;Умножим на число секторов в FAT
  add  accum,AX               ;Добавим в аккомулятор
  mov  AX,word ptr sysarea+11h   ;Число записей в корневом
                      ;каталоге(\)
  xor  DX,DX                  ;Будет деление слов
  mov  BX,16                  ;Число записей на сектор
  div  BX                    ;AX=число секторов в каталоге \
  add  accum,AX               ;Получился размер системной
                      ;облости в секторах
;Теперь прибавим число секторов в файле

  mov  AX,ES[DI]+50h          ;Текущий номер кластера
  sub  AX,2                  ;Нумерация кластеров с 2
  mov  BL,sysarea+0Dh          ;Число секторов кластере
  xor  BH,BH                  ;Будет умножение до нашего
  mul  BH                    ;кластера в области данных

  add  accum,AX               ;Логический сектор начала
                      ;последнего кластера файла
;Последний кластер уже прочитан только до конца файла,
;а ключ находится за его концом.Прочитаем кластер целиком

  mov  AL,3                   ;Дисковод Д:
```

```
  mov  CL,clustsize            ;число читаемых секторов
  xor  CH,CH
  mov  DX,accum                ;Логический номер сектора
  lea  BX.buffer               ;Буфер для чтения сектора
  int  25h                     ;Абсолютное чтение с диска
  pop  AX                      ;Подправим стек после INT 25h
;Прочитаем ключ из файла и сравним с ключем в програме
  mov  BX,endic                ;Число байтов до конца файла
  mov  AX,word ptr buffer[BX]    ;Прочитаем слово за
                               ;пределами файла
  cmp  AX,key                  ;Сравним с ключем в программе
  je   go3                     ;Равны!
  jmp  notins                  ;Не равны ,программа не установлена
;Программа устанолена можно с ней работать
;Выведем сообщение 'mes' об этом
 go3:
      ;Далее должна идти содержательная часть рабочей программы
      ;В данной задаче её может и не быть
                    . . .
   outprog:
;Завершим программу обычным образом
                    . . .
   notins:
;Вывод сообщения 'mes1' о том,что програма не установлена
                    . . .
   jmp  outprog
   notopen:
;Вывод сообщения 'mes2' о том,что файл не открылся
    jmp  outprog
   notfnd:
;Вывод сообщения 'mes3' о том,что файл не найден в таблице
;файлов
    jmp  outprog
;Подпрограмма сравнения имён файлов
                     . . .
;Поля данных
  key dw  1234h
next_blk  dd 0
fname     db   'work2.exe',0
workcon   db   'WORK2.EXE'
sysarea   db   512  dup (0)   ;Буфер для чтения сектора
accum     dw    0             ;Ячейка- аккомулятора
handle    dw    0   ;Ячейка для дескриптора файла WORK2.EXE
progsize  dw    0             ;Размер программы
endic     dw    0   ;Число байт файла в последнем кластере
```

```
clustsize db    0           ;Число секторов в кластере
clustbyte dw    0           ;Число байт в кластере
buffer    db    8*512 dup(0)
mes       db    'Программа установлена и будет работать нармально',10,13
meslen    equ   $-mes
mes1      db    'Программа не установлена',10,13
mes1len   equ   $-mes1
mes2      db    'Файл WORK2.EXE не открылся',10,13
mes2len   equ   $-mes2

mes3      db  'Файл WORK2.EXE не найден',10,13
mes3len   equ   $-mes3

;ПРОГРАММА INSTALL2.EXE

;Программа предназначенная для установки на жёстком диске
;рабочей программы WORK2.EXE
;Основные фрагменты программы
;Откроем файл WORK2.EXE с рабочей прог.,чтобы он
;попал в таблицу файлов
        . . .
;Получим доступ к таблице файлов
        . . .
;Обработаем таблицы файлов по очереди
;Сначала сохраним адрес следующей таблицы
        . . .
;И организуем поиск имени нашего файла в текушей таблице
        . . .
;Настроим ES:DI на следующий блок
        . . .
;Наконец наш файл найден
;Прочитаем с помощью прерывания Int 25h загрузочную запись
;нашего диска в буфер 'sysarea',чтобы определить размер
;кластера на данном жестком диске
 found:
            . . .
;Теперь получим двухсловную длинну программы и поделим
;на размер кластера
            . . .
;Для простоты не будем анализировать возможность того,что
;файл WORK2.EXE занемает целое число  кластеров без остатка
;Прочитаем последовательно все кластеры файла до последнего
           . . .
;Теперь в buffer последний кластер, а в таблице файлов-его номер
;По номеру кластера найдём логический номер
```

```
;его последнего сектора
;Сначала определим размер системной области
          . . .
;Теперь прибавим число секторов в файле
          . . .
;Последний кластер уже прочитан и находится в буфере buffer
;Запишем ключ сначала в буфер

  mov  BX,endic                ;Число байтов  файла в последнем
                          ;кластере
  cmp  AX,key                  ;Ключ
  mov  AX,word ptr buffer[BX]    ;Запишем ключ в буфер

;А затем и в файл
  mov  AL,3                   ;Дисковод Д:
  mov  CL,clustsize             ;число читаемых секторов
  xor  CH,CH
  mov  DX,accum                 ;Логический номер сектора
  lea  BX.buffer               ;Буфер для чтения сектора
  int  25h                    ;Абсолютное чтение с диска
  pop  AX                      ;Подправим стек после INT 25h

;Выведем сообщение 'mes' онармальной работе
                    . . .
   outprog:
;Завершим программу обычным образом
                    . . .
   notopen:
;Вывод сообщения 'mes2' о том,что файл не открылся
                    . . .
   jmp  outprog
   notfnd:
;Вывод сообщения 'mes3' о том,что файл не найден в таблице файлов
         . . .
    jmp  outprog
;Подпрограмма сравнения имён файлов
                     . . .
;Поля данных
next_blk  dd    0
fname     db   'work2.exe',0
workcon   db   'WORK2.EXE'
sysarea   db   512  dup (0)   ;Буфер для чтения сектора
accum     dw    0            ;Ячейка- аккомулятора
handle    dw    0   ;Ячейка для дескриптора файла WORK2.EXE
key       dw  1234h
```

```
progsize  dw    0           ;Размер программы
endic     dw    0   ;Число байт файла в последнем кластере
clustsize db    0           ;Число секторов в кластере
clustbyte dw    0           ;Число байт в кластере
buffer    db    8*512 dup(0)
mes       db    'Программа установлена на жёском диске',10,13
meslen    equ   $-mes
mes2      db   'Файл WORK2.EXE не открылся',10,13
mes2len   equ   $-mes2
mes3      db   'Файл WORK2.EXE не найден в таблице'файлов,10,13
mes3len   equ   $-mes3
```

Задача 3. Защита программ от копирования с помощью ключевой дискеты с нестандартным форматированием.

Написать программу, форматирующую дополнительную 41-ю дорожку дискеты (с номером 40) так, чтобы на ней были сектора нестандартного размера (например, не 512, а 256 байт). Записать в первый сектор этой нестандартной дорожки некоторый ключ.

Написать программу, которая перед выполнением считывает ключ с нестандартной дорожки и анализирует его правильность.

Программа INSTALL3.EXE

```
;Программа нестандартного форматирования дискеты и записи на нее
;ключа
;Основные фрагменты программы
;Установим тип дискеты (только для машин типа AT)
    mov   AH,17h
    mov   AL,02
    mov   DL,0
    int   13h
;Найдем и сохраним адрес таблицы параметров дискеты (вектор преры-
;вания 1Eh)
    mov   AX,351Eh
    int   21h
    mov   word ptr dpt,BX
    mov   word ptr dpt+2,ES
;Установим новый размер сектора 256 байт вместо 512
    mov   byte ptr ES:[DI+3],1
;Отформатируем дополнительнуб 41-ю дорожку
    mov   AH,05
    mov   CH,40
    mov   DH,0
    mov   DL,0
    push  DS
    pop   ES
```

```
    lea   BX,afd
    int   13h
;Записываем на дорожку ключ (размер min в целый сектор)
    mov   AH,03
    mov   AL,1
    mov   CH,40
    mov   CL,1
    mov   DH,0
    mov   DL,0
    lea   BX,key
    int   13h
;Восстановим таблицу параметров дискеты
go:   mov   ES,word ptr dpt+2,BX
    mov   byte ptr ES:[DI+3],2
;Поля данных
dpt   dd    0
afd   db    40,0,1,1
    db    40,0,2,1
    db    40,0,3,1
    db    40,0,4,1
    db    40,0,5,1
    db    40,0,6,1
    db    40,0,7,1
    db    40,0,8,1
    db    40,0,9,1
key   dw    9999h
    db    254 dup (0)
```

Программа WORK3.EXE

```
;Программа чтения нестандартно отформатированной дискеты и записи
;Основные фрагменты программы
;Установим тип дискеты - только для AT
    . . .
;Найдем и сохраним адрес таблицы параметров дискеты (вектор типа 1Eh)
    . . .
;Установим размер сектора нестандартной дорожки
;Прочитаем первый сектор 41-й дорожки
   mov      AH,02          Функция чтения сектора;
   mov      AL,1           ;Один сектор
   mov      CH,40          ;Цилиндр
   mov      CL,1           ;Сектор
   mov      DN,0           ;Головка
   mov      DL,0           ;Дисковод A:
   push     DS
   pop      ES
```

```
    lea     BX,buf          ;Адрес буфера в ES:BX
    int     13h
    jc      noform          ;Не читается
    cmp     word ptr buf,9999h  ;Сравним ключи
    jne     nokey           ;Неправильный ключ
;Выведем сообщение mes о нормальной работе
     . . .
;Завершим программу обычным образом, восстановив сначала
;таблицу параметров дискеты
outprog: mov    ES, word pptr dpt+2
        mov     byte ptr ES :[DI+3],2
noform:
;Введем сообщение mes2 о несоответствии форматов
     jmp outprog
nokey:
; Выведем сообщение mes3 о несоответствии ключа ожидаемому
     . . .
     jmp outprog
;Поля данных
dpt         dd      0
buf         db      256   dup (0)
mes         db      'Ключевая дискета установлена.',10,13
            db      'Программа может работать',10,13
mes  en     equ     $ -mes
mes2        db      'Установленна НЕ ключевая дискета',10,13
mes2 en     equ     $-mes2
mes3        db      'На дискете неправильный ключ',10,13
mes3 en     equ     $-mes3
```

9. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОРГАНИЗАЦИИ МЕЖСЕССИОННОГО КОНТРОЛЯ ЗНАНИЙ МАГИСТРОВ

1. Межсессионная аттестация магистров проводится дважды в семестр на 7 и 13 неделях семестра.

2. Аттестационная оценка выставляется по результатам работы в семестре: выполнения лабораторных работ по графику, выполнения контрольных работ и посещений лекционных занятий.

3. Организация аттестации магистров, проводится в соответствии с положением АмГУ о курсовых, экзаменах и зачетах.

- 10. КАРТА КАДРОВОЙ ОБЕСПЕЧЕНОСТИ ДИСЦИПЛИНЫ

Лектор – доцент, канд.техн.наук, Бушманов А.В.

Руководитель лабораторных работ – доцент, канд.техн.наук, Бушманов А.В.

Бушманов Александр Вениаминович,
к.т.н., доцент.

Применение методов моделирования в исследованиях и проектировании
Учебно-методический комплекс дисциплины

Изд-во АмГУ. Подписано к печати ???. Формат ???. Усл. печ. л. ???, уч. - изд. л. ???. Тираж 100. Заказ ???.

2

4