Министерство науки и высшего образования Российской Федерации
Министерство науки, высшего образования и инноваций Кыргызской Республики

Межгосударственная образовательная организация высшего образования
Кыргызско-Российский Славянский университет имени первого Президента Российской Федерации Б.Н. Ельцина.
Политехнический институт
Высшая школа цифровых технологий

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ

по дисциплине

АРХИТЕКТУРА КОМПЬЮТЕРОВ

Уровень высшего образования БАКАЛАВРИАТ

Направление подготовки
01.03.02 - РФ, 510200 - КР, Прикладная математика и информатика
(код и наименование направления подготовки)

Квалификация
Бакалавр

Бишкек 2026

Фонд оценочных средств предназначен для контроля знаний обучающихся по направлению подготовки (специальности) Прикладная математика и информатика» по дисциплине

«**АРХИТЕКТУРА КОМПЬЮТЕРОВ** »

Наименование

Фонд оценочных средств рассмотрен и утвержден на заседании высшей школы

Цифровые технологии

наименование высшей школы

Протокол №1 от «02» 09. 2026г.
Директор высшей школы Цифровых технологий



Хмелева Ирина Владимировна

наименование кафедры *подпись* *расшифровка подписи*

Руководитель образовательной программы

Доулбекова Салтанат Байызбековна Доцент 

ФИО *должность* *подпись*

Исполнители:

Доцент Аширбаев Бейшембек Ыбышевич 

расшифровка подписи подпись

Профессор Керимбеков Акылбек Керимбекович 

расшифровка подписи подпись

Раздел 1. Перечень компетенций, с указанием этапов их формирования в процессе освоения образовательной программы

Формируемые компетенции	Планируемые результаты обучения по дисциплине, характеризующие этапы формирования компетенций	Виды оценочных средств/ шифр раздела в данном документе
ОПК-2. Способен использовать и адаптировать существующие математические методы и системы программирования для разработки и реализации алгоритмов решения прикладных задач.	**<u>Знать:</u>** - Способен использовать и адаптировать существующие математические методы операторные методы решения уравнений; - графические методы решения задач; - методы оптимизации и оптимального управления; - численные методы решения прикладных задач; - возможности аппаратных и программных средств; - особенности Системного ПО их функционирования в многозадачных ОС; - средства разработки приложений, их состав и функции; - организацию программного обеспечения, типовые методы организации и свойства компонент операционных систем, механизмы исполнения программ в операционных системах; - основные конструкции языков программирования и принципы разработки программ, динамические структуры данных, объектно- ориентированную методологию программирования.	Блок А: СРС – лабораторная работа Практическое задание
	<u>Уметь:</u> - разработать на основе аналитических методов алгоритм решения математических и прикладных задач для использования их на практике; - разработать алгоритм численных методов решения нелинейных прикладных задач для использования их в приложениях; - -использовать разработанные алгоритмы для решения прикладных задач естественных наук; - осуществлять обоснованный выбор средств программирования; - использовать изученные алгоритмы	Блок Б: тесты

	программирования для решения конкретных задач; - самостоятельно создавать и тестировать системные модули, используя передовые информационные технологии; - применять знания в разработке баз данных; - применять знания построения системных и обрабатывающих программ; - демонстрировать глубокие знания: - построения параллельных программ; - работать в среде разработки Microsoft Visual Studio.NET; - использовать регулярные выражения; - создавать простейшие Web Application Project с применением форм и проверкой данных при вводе их пользователем.	
	Владеть: -навыками применения аналитических и численных методов решения линейных, нелинейных и операторных уравнений при исследовании и решении прикладных задач и задач естественных наук; - навыками программирования и осуществлять поиск дополнительной информации для изучения тематических разделов по математике и информатике; -основными навыками работы в операционной среде WINDOWS; - получить основные навыки в работе с основными приложениями, входящими в состав пакета Microsoft Office; -навыками работы со стандартными средами современных операционных систем и разработки собственных приложений системного назначения; - методами и средствами программирования распределенных ВС и сетей; методами и средствами анализа и разработки программных компонентов сетевых и телекоммуникационных систем; - понятия о нормализации баз данных; - навыками работы со стандартными средами современных операционных систем - навыками разработки собственных параллельных приложений.	

	- навыками работы в среде разработки Microsoft Visual Studio.NET; - навыками создании, отладке, компиляции и выполнении программы на C#; - навыками использования справочной системы среды программирования .NET	
ОПК-4. Способен понимать принципы работы современных информационных технологий и использовать их для решения задач профессиональной деятельности	**Знать:** - архитектуру вычислительных сетей, технологии распределенной обработки, сетевые технологии; - методы и средства проектирования программных средств, компоненты системного программного обеспечения и технологии их разработки; - основные направления современных информационных технологий.	
	Уметь: - выбирать методы решения, соответствующие задачам профессиональной деятельности; Использовать современные информационные технологии для решения задач; - интерпретировать полученные результаты.	
	Владеть: - теоретическими знаниями для разработки программ с использованием современных технологий; - Навыками проведения анализа и принятия решений, согласно задачам профессиональной деятельности.	

Раздел 2. Технологическая карта дисциплины

Технологическая карта дисциплины

«Архитектура компьютеров»

Курс/семестр: 2/3

Количество кредитов(ЗЕ): 5

Отчетность: Зачет с оценкой

Название модулей дисциплины согласно РПД	Контроль	Форма контроля	зачетный минимум	зачетный максимум	график контроля
Модуль 1					

Представление информации в компьютере.	Текущий контроль	Групповая дискуссия, тестирование	6	8	3
	Рубежный контроль	Практическое занятие, индивидуальные задания	5	8	
Модуль 2					
Архитектура компьютера	Текущий контроль	Групповая дискуссия, тестирование	3	6	5
	Рубежный контроль	Практическое занятие, индивидуальные задания	2	6	
Модуль 3					
Архитектура микропроцессора	Текущий контроль	Групповая дискуссия, тестирование	3	6	7
	Рубежный контроль	Практическое занятие, индивидуальные задания	4	6	
Модуль 4					
Введение в ассемблер	Текущий контроль	Групповая дискуссия, тестирование	5	6	10
	Рубежный контроль	Практическое занятие, индивидуальные задания	4	6	
Модуль 5					
Программирование на ассемблере	Текущий контроль	Групповая дискуссия, тестирование	3	10	12
	Рубежный контроль	Практическое занятие, индивидуальные задания	5	8	
ВСЕГО за семестр			40	70	
Промежуточный контроль (Экзамен)		тест	20	30	
Семестровый рейтинг по дисциплине			60	100	

Модуль	логически завершенная часть дисциплины
Текущий контроль	самостоятельная работа студента, посещаемость и активность на занятиях

Рубежный контроль	проверка полноты знаний и умений по материалу модуля в целом
Промежуточный контроль	завершенная задокументированная часть учебной дисциплины - совокупность тесно связанных между собой модулей дисциплины.

Раздел 3. Типовые контрольные задания и иные материалы, необходимые для оценки планируемых результатов обучения по дисциплине. Описание показателей и критериев оценивания компетенций, описание шкал оценивания.

Блок А: Практическое задание

Задание

Составить отчет по практической работе, используя приведенный шаблон.

1. Определить состав ПК и составить ее техническую характеристику подобно приведенной ниже.
2. Краткую характеристику ПЭВМ составить, используя программы:
   a. из группы Стандартные ->.Служебные->Сведения о системе
   b. программу AIDA32.
3. Составить структурную схему ПК, указав основные характеристики ее составных частей. Используйте приведенную в задании схему для ее модификации.
4. Определить состав системного программного обеспечения ПК и составить схему их взаимодействия.

Отчет по работе

1. Состав ПК на основе прилагаемой документации

Центральный процессор:
Системная плата:
Оперативная память:
Разъемы:
Кэш - память:
Разъемы расширения:
Параллельные порты:
Последовательные порты:
Разъемы для подключения дисковых накопителей:
Дисковые накопители:
Видеоадаптер:

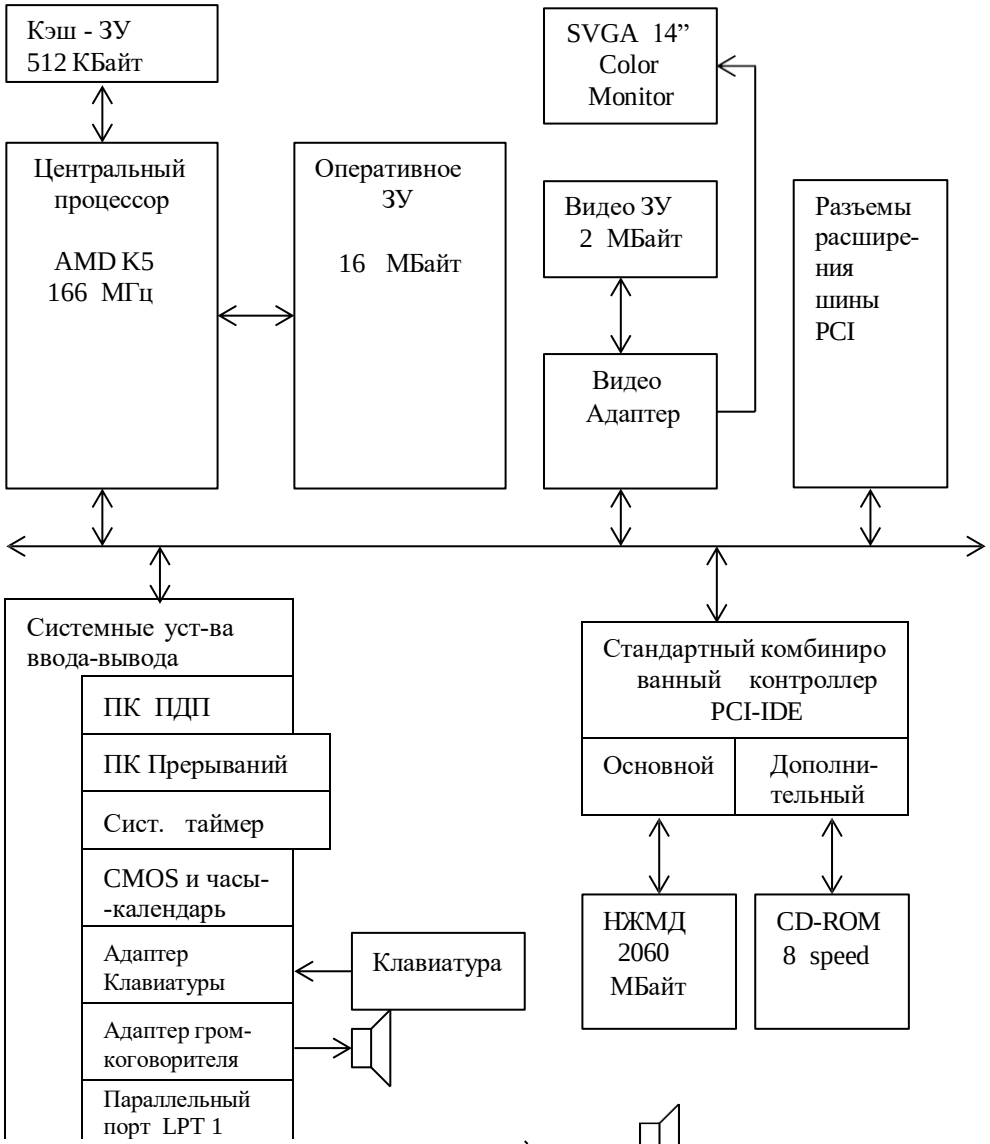
Монитор:
Звуковая плата:
Мышь:
Клавиатура:

2. ***Краткая характеристика ПК, полученная с использованием ее программных средств***

Центральный процессор:
Системная плата:
Оперативная память:
Кэш - память:
BIOS :
Параллельные порты:

Последовательные порты:
Накопитель на жестком магнитном диске:
Устройство чтения компакт-дисков:
Накопитель на гибком магнитном диске:
Видеоадаптер:
Монитор:
Звуковая плата:
Мышь :
Клавиатура:

3. ***Структурная схема ПК***



4. Состав системного программного обеспечения ПК.

Используйте ниже приведенную схему программного обеспечения для определения состава системного программного обеспечения персонального компьютера.

Тема:

Работа с машинными командами и командами ассемблера с помощью отладчика DEBUG

Цель работы

Знакомство с отладчиком debug.exe. Получение практических навыков работы с данной программой.

Программное обеспечение:

Программа Debug.

Задание:

1. Изучить **теоретический материал,** проделав поэтапно описанные по ходу текста упражнения. Затем выполнить приведенные ниже задания.
2. Найдите сумму и разность 2-х чисел: 1-е число – количество букв в вашей фамилии+ количество букв в вашем имени (переведенное в шестнадцатиричную форму), 2-е – ваш день рождения. Сумму и разность переведите вручную в десятичную форму, результат и само решение напишите в отчете.
3. Просмотрите содержимое регистров микропроцессора, а также флагов и вставьте в отчет в виде скриншота. Объясните какую функцию выполняет каждый из регистров?

Диапазон области памяти задайте используя для начала диапазона порядковый номер в журнале студенческой группы, для конца диапазона количество букв в вашей фамилии. Например студент Алексеев имеет порядковый номер в журнале 1, вводим D 0001:0008.

4. Запишите в регистр AX первое число (из задания 2), а в регистр BX - второе (из задания 2). Введите в оперативную память в сегмент кода (смещение 100) машинную команду сложения регистров AX и BX. Просмотрите на экране ее ассемблерную форму. Выполните эту команду, результат переведите в десятичную форму.
5. Введите в оперативную память в сегмент кода (смещение 100) набор команд ассемблера для распечатки символа на экране - первой буквы вашей фамилии. Проверьте программу в DEBUG. Затем запишите ее на диск в виде .COM-файла. Чему равен размер программы? Запустите ее на выполнение из DOS.
6. Все действия опишите в отчете.

Содержание отчета:

1. Название работы.
2. Цель работы.
3. Приборы и оборудование.
4. Краткие теоретические сведения.
5. Описание проделанных действий. (**привести скриншоты**).
6. Текст созданной программы.
7. Ответы на контрольные вопросы.

Контрольные вопросы

1. Что называется смещением сегмента?
2. Какой формат имеют машинные команды МП?
3. Как называется программа, позволяющая записывать коды чисел и команд в ячейки памяти?
4. Как осуществляется запуск программы Debug в среде ОС Windiws?
5. Содержимое каких регистров МП используется для формирования адреса ячейки сегментов кодов?
6. Как можно вывести содержимое всех регистров МП на экран монитора?
7. Укажите порядок изменения содержимого регистра ip.
8. Укажите порядок изменения содержимого регистра ax.
9. Что означает выражение: «выполнение программы в режиме трассировки»?
10. Какая команда программы выполняется при вводе команды и программы Debug.
11. Какая команда производит ввод в память данные или инструкции машинного кода?
12. Как сравнить содержимое двух областей памяти?
13. Каким образом заполнить область памяти данными из списка?
14. Какая команда производит выполнение отлаженной программы на машинном языке?
15. Какая команда записывает файл из Debug?
16. Как производится ассемблирование и дизассемблирование?

Краткие теоретические сведения

Программа DEBUG используется для тестирования и отладки исполняемых программ. Программа DEBUG показывает код и данные программы в шестнадцатеричном формате, и любые данные, которые вводятся в память, также должны быть в этом формате. DEBUG также реализует пошаговый режим исполнения, позволяющий выполнять инструкции программы по отдельности одну за другой и наблюдать результат выполнения каждой инструкции в памяти и регистрах.

Команды DEBUG

Таблица 1. Краткая таблица всех команд debug.exe

Команда	Описание	Формат
A (Assemble)	Транслирование команд ассемблера в машинный код; адрес по умолчанию - CS:0100h.	A [<адрес_начала_кода>]
C (Compare)	Сравнение содержимого двух областей памяти; по умолчанию используется DS. В команде указывается либо длина участков, либо диапазон адресов.	C <начальный_адрес_1> L<длина> <начальный_адрес_2> C <начальный_адрес_1> <конечный_адрес_1> <начальный_адрес_2>
D (Display/Dump)	Вывод содержимого области памяти в шестнадцатеричном и ASCII-форматах. По умолчанию используется DS; можно указывать длину или диапазон.	D [<начальный_адрес> [L<длина>]] D [начальный_адрес конечный_адрес]
E (Enter)	Ввод в память данные или инструкции машинного кода; по умолчанию используется DS.	E [<адрес> [<инструкции/данные>]]
F (Fill)	Заполнение области памяти данными из списка; по умолчанию используется DS. Использовать можно как длину, так и диапазон.	F <начальный_адрес_1> L<длина> '<данные>' F <начальный_адрес> <конечный_адрес> '<данные>'
G (Go)	Выполнение отлаженной программы на машинном языке до указанной точки останова; по умолчанию используется CS. При этом убедитесь, что IP содержит корректный адрес.	G [=<начальный_адрес>] <адрес_останова> [<адрес_останова> ...]
H (Hexadecimal)	Вычисление суммы и разности двух шестнадцатеричных величин.	H <величина_1> <величина_2>
I (Input)	Считывание и вывод одного байта из порта.	I <адрес_порта>
L (Load)	Загрузка файла или данных из секторов диска в память; по умолчанию - CS:100h. Файл можно указать с помощью команды N или аргумента при запуске debug.exe.	L [<адрес_в_памяти_для_загрузки>] L [<адрес_в_памяти_для_загрузки> [<номер_диска> <начальный_сектор> <количество_секторов>]]
M (Move)	Копирование содержимого ячеек памяти; по умолчанию используется DS.	M <начальный_адрес> L<длина> <адрес_назначения> M <начальный_адрес> <конечный_адрес> <адрес_назначения>

	Можно указывать как длину, так и диапазон.	
N (Name)	Указание имени файла для команд L и W.	N <имя_файла>
O (Output)	Отсылка байта в порт.	O <адрес_порта> <байт>
P (Proceed)	Выполнение инструкций CALL, LOOP, INT или повторяемой строковой инструкции с префиксами REPnn, переходя к следующей инструкции.	P [=<адрес_начала>] [<количество_инструкций>]
Q (Quit)	Завершение работы debug.exe.	Q
R (Register)	Вывод содержимого регистров и следующей инструкции.	R <имя_регистра>
S (Search)	Поиск в памяти символов из списка; по умолчанию используется DS. Можно указывать как длину, так и диапазон.	S <начальный_адрес> L<длина> '<данные>' S <начальный_адрес> <конечный_адрес> '<данные>'
T (Trace)	Пошаговое выполнение программы. Как и в команде P, по умолчанию используется пара CS:IP. Замечу, что для выполнения прерываний лучше пользоваться командой P.	T [=<адрес_начала>] [<количество_выполняемых_команд>]
U (Unassemble)	Дизассемблирование машинного кода; по умолчанию используется пара CS:IP. К сожалению, debug.exe некорректно дизассемблирует специфические команды процессоров 80286+, хотя они все равно выполняются корректно.	U [<начальный_адрес>] U [<начальный_адрес конечный_адрес>]
W (Write)	Запись файла из debug.exe; необходимо обязательно задать имя файла командой N, если он не был загружен. А программы записываются только в виде файлов .COM!	W [<адрес> [<номер_диска> <начальный_сектор> <количество_секторов>]]

Основные особенности программы DEBUG.

- DEBUG не различает строчные и заглавные буквы.
- Все вводимые числа задаются в шестнадцатеричной форме.
- Пробелы используются в командах только для разделения параметров.

• Сегмент и смещение указываются с использованием символа двоеточия, т.е. в форме сегмент:смещение.

После загрузки отладчика на экране появится приглашение, выглядящее в виде дефиса. Регистры CS, DS, ES, SS в этот момент инициализированы адресом 256-байтного префикса сегмента программы, а рабочая область в памяти будет начинаться с адреса этого префикса + 100h. Команды debug.exe вводятся сразу после приглашения на месте, которое отмечено курсором. Каждая команда состоит из идентификатора и параметров, идентификатор состоит из одной буквы.

Команда D (Display - показать) в программе DEBUG

Эта команда выводит содержимое указанной области памяти на экран. Следующие три примера используют команду D для просмотра одного и того же участка памяти, начинающегося со смещения 3C1H в сегменте кода (CS):

D CS:3C1 (команда набрана заглавными буквами с пробелом);

DCS:3C1 (команда набрана заглавными буквами без пробела);

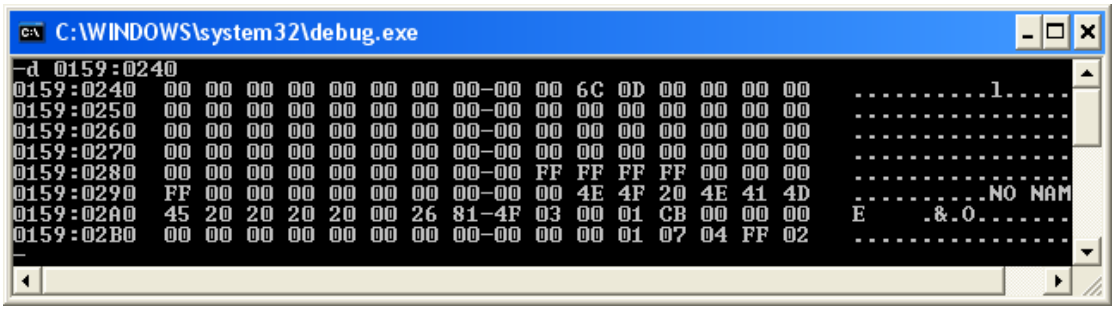
dcs:3cl (команда набрана строчными буквами без пробела);

В результате выполнения команды получается восемь строк. В левой части каждой из них будет указан адрес первого слева показанного байта в форме сегмент: смещение. Основную часть строки, в центре, занимает шестнадцатеричное представление параграфа, начинающегося с указанного в начале строки байта. Справа на экран выведены в ASCII-форме символы этого параграфа, которые можно вывести на экран, для облегчения интерпретации шестнадцатеричной записи.

Команда D выводит 8 строк данных, в каждой из которых по 16 байт (32 шестнадцатеричных разряда), всего – 128 байт, начиная с указанного адреса.

Адрес слева относится только к первому байту в строке, адреса последующих байтов могут быть легко найдены простым счетом: например, если адрес первого байта – 0159:0240H, то одиннадцатый байт в строке имеет адрес 0159:024AH. Шестнадцатеричное представление содержит два знака для каждого байта, байты разделяются пробелами для улучшения читаемости. Кроме того, восьмой и девятый байты разделяет дефис. Поэтому, например, если требуется найти байт со смещением xx13H, начните с байта xx10H и найдите третий после него байт.

Команда D также показывает содержимое регистров и состояние флагов в регистре Flags.



Упражнение 1: просмотр области данных BIOS

Первое упражнение показывает содержимое области данных BIOS в памяти, начиная с адреса 400H или, более точно, с адреса сегмента 40[0]H. BIOS инициализирует значения в этой области памяти при включении компьютера и меняет их в ходе выполнения программ.

Просматривайте эти значения при помощи адреса из двух частей: в качестве адреса сегмента (то есть 400, с отброшенным младшим разрядом), и nn в качестве смещения от начала сегмента. Воспринимайте адрес 40:nn как сегмент 40[0]H плюс смещение nnH.

Проверка параллельных и последовательных портов

Первые 16 байт области данных BIOS содержат адреса параллельных и последовательных портов. Введите следующую команду:

D 40:00 (и нажмите <Enter>)

Первые четыре выведенных слова указывают на адреса портов от COM1 до COM4. Если на компьютере два последовательных порта, первые два слова, вероятно, содержат F803 и F802 в обращенной (с переставленными байтами) последовательности. Последовательные порты имеют адреса 03F8 и 02F8. Следующие 4 слова указывают на параллельные порты от LPT1 до LPT4. Для системы с одним параллельным портом первое слово, вероятно, содержит 7803, т.е. адрес порта – 0378.

Проверка оборудования системы

Слово состояния оборудования в области данных BIOS предлагает базовую информацию о присутствующих в системе устройствах. Это слово, расположенное по адресу 410H - 411H, можно просмотреть командой

D 40:10 (и нажмите <Enter>)

Выведенные строки должны начинаться так:

0040:0010 xx xx ...

Допустим, что слово содержит 23 44 в шестнадцатеричной форме. Чтобы интерпретировать его, переставим байты (44 23) и преобразуем в двоичную форму:

Двоичное значение:	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	1
Позиция бита:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Эти биты означают:

Биты	Устройство
15, 14	Число параллельных (принтерных) портов = 1 (двоичное 01)
11 – 9	Число последовательных портов = 2 (двоичное 010)
7,6	Число дисководов = 1 (00 = 1,01 = 2, 10 = 3, 11 = 4)
5,4	Начальный видеорежим = 10 (01 = 40x25 цветной, 10 = 80x25 цветной, 11 = 80x25 монохромный)
1	1 = присутствует математический сопроцессор
0	1 = присутствует привод для дискет

Неописанные биты не используются.

Проверка состояния регистра клавиатуры (клавиша Shift)

В области данных BIOS по адресу 417H хранится первый байт состояния регистра клавиатуры. Убедитесь, что Num Lock и Caps Lock выключены, и просмотрите содержимое байта по этому адресу командой D 40:17.

Результат будет похож на

0040:0017 00 00 ...

Теперь включите Num Lock и Caps Lock и введите вновь команду D 40:17. Результат должен начинаться с 60 00.

Проверка состояния видеоустройства

В области данных BIOS по адресу 449H находится первая область видеоданных (Video Data Area). Введите команду D 40:49. Первый показанный байт указывает на

текущий видеорежим (например, 03 - цветной), а второй – число столбцов на экране (например, 50 означает режим с 80 столбцами). Число строк хранится по адресу 40:84H.

Упражнение 2: Просмотр ROM BIOS

Сведения об авторском праве BIOS системы встроены в ROM BIOS no адресу FE00:0. В зависимости от производителя компьютера будут выведены различные строки, после которых будет указан семизначный серийный номер. Строка, указывающая на авторство BIOS, легко читается в виде ASCII-последовательности, а серийный номер – в виде шестнадцатеричного числа. Строка с указанием авторских прав может быть длиннее, чем показанный участок памяти; в этом случае для просмотра не показанной части снова введите D и нажмите <Enter>.

Проверка даты производства BIOS

Эта дата, записанная в виде мм/дд/гг, начинается с адреса FFFF5H. Для ее просмотра введите команду D FFFF:5. Знание этой даты полезно при определении возраста и модели компьютера.

Используя команду D, вы можете просмотреть содержимое любой области памяти. Можно также последовательно просматривать память, просто повторно вводя D – DEBUG будет выводить 128 байт, следующих за последними просмотренными.

Для завершения работы с DEBUG введите Q.

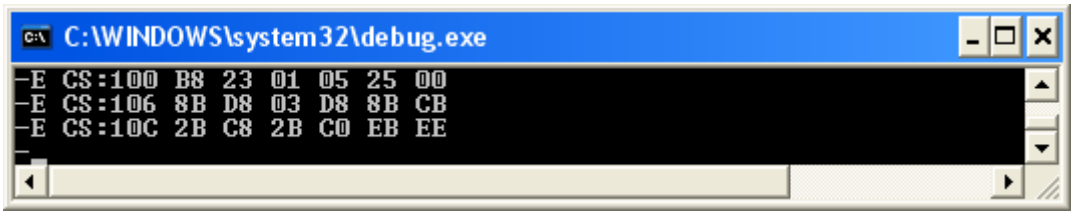
Упражнение 3: Использование непосредственных данных

Воспользуемся DEBUG для ввода первой из двух программ непосредственно в память и наблюдения за ее выполнением. Обе программы включают простые инструкции машинного языка в виде, в котором они находятся в памяти, и позволяют продемонстрировать эффект их выполнения. Выполнение упражнения начинается с команды E (Enter, ввести).

Первая программа использует непосредственные данные - данные, определенные в теле инструкций. Далее показан как машинный код и соответствующий символьный код с комментариями для улучшения восприятия. Первой инструкции, B82301, соответствует символьный код MOV AX,0123, заносящий (копирующий) значение 0123H в регистр AX (непосредственные данные записываются в прямой, с непереставленными байтами, форме). MOV – инструкция, AX – ее первый операнд, непосредственное значение 0123H – второй.

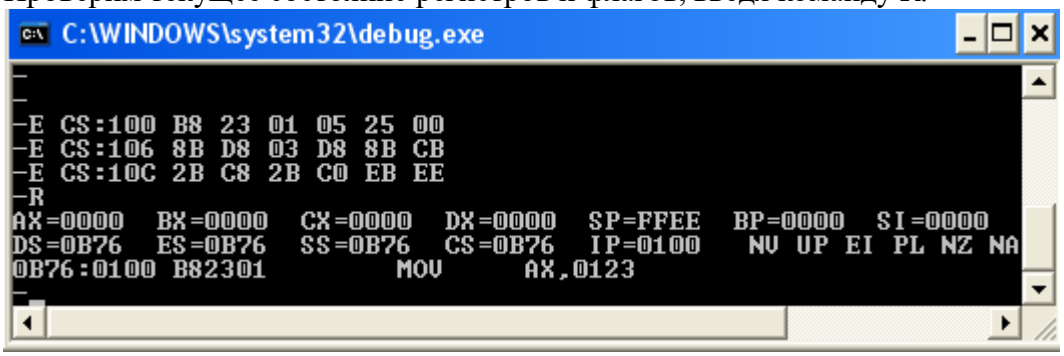
```
Машинная        Символьный код          Пояснение
инструкция

 B82301          MOV AX,0123     Переслать значение 0123h в AX
 052500          ADD AX,0025     Прибавить значение 0025h к AX
    8BD8            MOV BX,AX      Переслать содержимое AX в BX
    03D8            ADD BX,AX      Прибавить содержимое AX к BX
    8BCB            MOV CX,BX      Переслать содержимое BX в CX
    2BC8            SUB CX,AX      Вычесть содержимое AX из CX
    2BC0            SUB AX,AX      Вычесть содержимое AX из AX (очистка AX)
    90              NOP            Нет операции
    EBEE            JMP 100        Переход к началу программы.
```

Из примера видно, что машинные инструкции имеют длину 1, 2 или 3 байта. Первый байт указывает, собственно, операцию, а все последующие – ее операнды (непосредственные значения, ссылки на регистры или адреса в памяти). Исполнение программы начинается с первой машинной инструкции и последовательно проходит через все инструкции одну за другой.



В примере машинный код разделен на три части по шесть байт и каждая вводится, используя команду E и начиная с адреса CS:100.

Проверим текущее состояние регистров и флагов, введя команду R.



Итак, как можно видеть, debug.exe инициализировал сегменты DS, ES, SS, CS одним и тем же адресом. Регистр IP содержит 0100, указывая на то, что инструкции выполняются со смещения 100h относительно CS (а мы, вводя инструкции в память, как раз указали этот адрес).

Здесь же указаны и значения флагов переполнения, направления, прерывания, знака, нуля, дополнительного переноса, четности и переноса:

Значение	**Описание**
NV	Отсутствие переполнения
UP	Направление вверх или вправо
EI	Разрешение прерываний
PL	Положительный знак
NZ	Ненулевое значение
NA	Отсутствие дополнительного переноса
PO	Нечетное слово
NC	Отсутствие переноса

Ввод инструкций программы

Для того чтобы ввести свою программу непосредственно в память, вводите только машинный код, символьный код и комментарии водить не нужно. Введите команду E и следующие символы, как показано ниже.

E CS:100 B8 23 01 05 25 00 (и нажмите <Enter>)

CS:100 указывает начальный адрес области, в которой будет храниться введенный код (обычный начальный адрес для машинного кода в программе DEBUG). Команда E сохранит следующие за ней байты в адресах от CS:100 до CS:105.

Следующая команда E сохранит еще 6 байт в адресах от CS:106 до CS:10B.

E CS:106 8B D8 03 D8 8B CB (нажмите <Enter>)

Последняя команда E сохраните 6 байт в адресах от CS:10C до CS:111.

E CS:10C 2B C8 2B C0 EB EE (нажмите <Enter>)

Если вы введете неправильную строку, просто повторите ввод правильно.

Выполнение инструкций программы

Теперь можно выполнить введенные инструкции по очереди. Можно также просматривать содержимое регистров после выполнения каждой инструкции с помощью команд R (registers - регистры) и T (trace - отследить).

Для просмотра начального состояния регистров и флагов введите команду R и нажмите <Enter>. DEBUG показывает содержимое регистров в шестнадцатеричной форме в виде

AX=0000 BX=0000 ...

DEBUG проинициализировал DS, CS, SS и ES одним и тем же адресом, xxxx[0]. IP должен показывать 0100, указывая, что выполнение инструкций начинается со смещения 100H относительно начала сегмента кода.

Регистр флагов показывает следующие значения флагов переполнения, направления, прерывания, знака, нуля, дополнительного переноса, четности и переноса:

NV UP El PL NZ NA PO NC

Эти значения соответствуют слева направо: отсутствию переполнения, направлению вверх (или вправо), разрешению прерываний, положительному знаку, ненулевому значению, отсутствию дополнительного переноса, нечетному слову, и отсутствию переноса. В данный момент эти значения не важны.

Сразу после регистров командой R показана первая, подлежащая выполнению инструкция:

xxxx:0100 B82301 MOV AX,0123

- xxxx означает начало сегмента кода по адресу xxxx[0]H. Значение xxxx:0100 означает смещение 100H от начала сегмента кода, начинающегося по адресу xxxx[0]H.
- B82301 – машинный код, введенный по адресу CS:100.
- MOV AX,0123 – символьная инструкция на языке Ассемблера, определенная DEBUG из машинного кода. Эта инструкция перемещает (копирует) непосредственное значение 0123H в регистр AX. DEBUG дизассемблировал машинный код для лучшего его понимания.

Для выполнения инструкции MOV введите команду T и нажмите <Enter>. Машинный код команды – B8 (поместить в AX), операнд – 2301. Операция помещает 23 в младшую половину регистра (AL) и 01 – в старшую (AH).

DEBUG выводит результаты выполнения – эффект, оказанный операцией на содержимое регистров. Регистр IP содержит теперь 0103H (0100H плюс 3 байта – длина выполненной инструкции). Это значение указывает на адрес, по которому расположена следующая подлежащая выполнению инструкция, то есть:

```
xxxx:0103 052500  ADD AX,0025
```

Для выполнения этой инструкции вновь введите T. Инструкция добавит 25H к младшему байту (AL) регистра AX и 00H – к старшему (AH), то есть добавит 0025H к содержимому AX. Теперь AX содержит 0148H, а IP – 0106H, указывая на следующую инструкцию, которую необходимо выполнить:

```
xxxx:0106 8BD8  MOV AX,BX
```

Вновь введите команду T. Инструкция MOV заносит содержимое регистра AX в регистр BX. После выполнения инструкции BX содержит 0148H. AX все еще хранит 0148H, поскольку инструкция MOV копирует данные, не удаляя их из источника.

Теперь последовательно вводите команды T для выполнения оставшихся инструкций. Инструкция ADD суммирует содержимое регистров AX и BX, записывая значение 0290H в регистр BX. Затем программа копирует содержимое BX в CX, вычитает AX из CX и вычитает AX из самого себя. Последняя инструкция очищает AX, устанавливая

его в 0, и устанавливает флаг ZF из NZ (nonzero – не нуль) в ZR (zero – нуль), отображая результат операции.

Инструкция JMP устанавливает IP в 100H, и обработка возвращается обратно к началу программы. Эта инструкция введена из соображений предосторожности, поскольку за последней введенной инструкцией следует "мусор", который при попытке выполнения мог бы вызвать останов процессора или другие нежелательные результаты.

Регистры DS, ES, SS и CS содержат один и тот же адрес сегмента. Это потому, что DEBUG рассматривает весь введенный код как программу .COM, хранящую данные, стек и код в одном сегменте, хотя вы и храните их по отдельности внутри сегмента. При написании программы .EXE стек, данные и код хранятся в отдельных сегментах с разными адресами.

Для повторного выполнения программы введите еще раз команду T, DEBUG выполнит инструкцию JMP и перейдет к началу введенной программы.

Просмотр содержимого памяти

Для просмотра программы на машинном языке в сегменте кода запросите вывод информации командой D CS:100. Результаты выполнения этой команды представляются строками с 16 байтами (32 шестнадцатеричными разрядами) в каждой строке. Справа – ASCII представление (если соответствующий символ можно вывести) каждого байта. В случае машинного кода ASCII представление бессмысленно и может игнорироваться.

Первая строка начинается со смещения 100H в сегменте кода и содержит байты с адресами от CS:100H до CS:10FH. Вторая строка выводит байты с адресами от CS:110H до CS:11FH. Хотя введенная программа заканчивается байтом с адресом CS:111H, DEBUG автоматически выводит восемь строк – от CS:100H до CS:170H. В этом примере все данные, следующие за CS:111H – "мусор".

Введите Q для завершения работы с DEBUG или начните выполнение следующего упражнения.

Использование ранее определенных данных

Предыдущий пример использовал непосредственные данные, определенные прямо в директивах MOV и ADD. В этом примере данные (числовые константы 0123H и 0025H) определены в виде отдельных элементов данных в программе. Инструкции программы должны работать с ячейками памяти, содержащими эти значения.

Этот пример показывает, как компьютер получает доступ к данным с помощью адреса в регистре DS и смещений. В примере, начиная с адреса DS:0200H, определены следующие элементы данных.

Смещение в сегменте DS	Шестнадцатеричный код содержимого
0200H	2301H
0202H	2500H
0204H	0000H
0206H	2A2A2AH

Шестнадцатеричная цифра занимает полбайта, поэтому, например, 23H хранится в байте с адресом 200H, а 01H – в следующем (со смещением 201H). Ниже приведены машинные инструкции, обрабатывающие эти данные, со значениями, введенными в обратной (с переставленными байтами) форме.

Инструкция	Объяснение
A10002	Поместить слово (2 байта), начинающееся с адреса DS:0200H, в регистр AX
03060202	Добавить содержимое слова (2 байта), начинающегося с адреса DS:0202H, к содержимому регистра AX

A30402	Поместить содержимое AX в слово, начинающееся с адреса DS:0204H
EBF4	Перейти к началу программы

Две инструкции по перемещению данных имеют разные машинные коды: A1 и A3. Действительный машинный код инструкции MOV зависит от регистров, на которые ссылается инструкция, размера элемента данных (байт или слово), направления передачи данных (в регистр или из регистра) и того, ссылается инструкция на непосредственные данные, адрес в памяти или регистр.

Ввод инструкций и данных программы

Используйте программу DEBUG для ввода и выполнения программы этого примера. Сначала с помощью команды E введите инструкции, начиная с адреса CS:0100:

E CS:100 A1 00 02 03 06 02 02 (и нажмите <Enter>)

E CS:107 A3 04 02 EB F4 (и нажмите <Enter>)

Теперь (также при помощи команды E) введите соответствующие значения в ячейки сегмента данных:

```
E DS:0200 23 01 25 00 00 00 (и нажмите <Enter>)
E DS:0206 2A 2A 2A (и нажмите <Enter>)
```

Первая команда E сохраняет три слова (6 байт) в начале области данных – начиная со смещения 0200H. Все эти слова должны быть введены в форме с переставленными байтами, то есть 0123 как 2301 и 0025 как 2500. Когда в дальнейшем инструкция MOV запросит эти слова из памяти, байты будут переставлены опять, образуя исходные значения.

Вторая команда E вводит в память три символа звездочки (***) в виде кода 2A2A2A. Вы сможете их увидеть в виде символов с помощью команды D.

Для просмотра данных в сегменте (со смещениями от 0200H до 0208H) и инструкций в сегменте CS (от 0100H до 010AH) введите следующие команды D:

Для просмотра кода: 100,10A <Enter>

Для просмотра данных: 200,208 <Enter>

Выполнение инструкций программы

Введя инструкции, можно выполнять их тем же способом, что и в предыдущем примере. Сначала убедитесь, что IP содержит 0100H. После этого командой R просмотрите содержимое регистров, флагов и код первой инструкции. Хотя AX может все еще содержать значение, присвоенное ему в предыдущем примере, оно скоро будет заменено новым. Первая показанная инструкция

xxxx:0100 A10002 M0V AX,[0200]

CS:0100 указывает на первую инструкцию, A10002. DEBUG интерпретирует эту инструкцию как MOV и определяет, что она ссылается на ячейку со смещением [0200H] в сегменте DS. Квадратные скобки указывают, что это не непосредственное значение, а адрес в памяти. (Непосредственное значение для записи в AX выглядело бы как MOV AX,0200.)

Теперь введите команду T. Инструкция выполняется и помещает значение слова со смещения 0200H в регистр AX. Содержимое этого слова – 2301H, а инструкция переставляет байты и помещает его в регистр AX в виде 0123H, стирая любое предыдущее значение.

Введите вторую команду T. Будет выполнена следующая инструкция, т.е. ADD. Операция добавляет значение из слова по адресу DS:0202H к значению регистра AX. Результат – сумма 0123H и 0025H, то есть 0148H.

Следующая инструкция – MOV [0204],AX. Введите T для ее исполнения. Инструкция копирует значение из регистра AX в слово в памяти, занимающее ячейки с адресами 0204H и 0205H. При этом байты будут переставлены, и слово будет содержать значение 4801H. Для просмотра изменившегося содержимого ячеек памяти введите команду

```
D DS:200,208  <Enter>
```

Выведенные значения должны быть такими:

```
Значение в ячейке: 23   01   25   00   48   01   2A   2A   2A
Смещение:         200  201  202  203  204  205  206  207  208
```

Левая сторона дисплея показывает действительный машинный код так, как он хранится в памяти. Правая сторона помогает легко находить символьные данные. Эти шестнадцатеричные значения представлены в правой части экрана соответствующими им символами ASCII. Коды 23H и 25H выводятся как # (символ номера) и % (символ процента) соответственно. Три байта 2AH выводятся как три звездочки (*).

Можно завершить работу с DEBUG командой Q или перейти к выполнению следующего примера.

Повторное выполнение инструкций

1. Иногда необходимо вручную установить значение в регистре IP. Делается это так.
2. Введите команду R IP для вывода содержимого регистра IP.
3. Введите значение 100 (или адрес другой инструкции), а затем нажмите <Enter>.

Эта процедура возвращает к началу программы (или к инструкции внутри программы), и можно повторно выполнить уже пройденные шаги. Введите команду R (без IP). DEBUG выведет на экран содержимое регистров, флаги и следующую подлежащую выполнению инструкцию. Теперь можно использовать команду T для повторного выполнения инструкций. Если программа накапливает значения, используйте команду E для очистки ячеек памяти и команду R для очистки регистров.

Сохранение программы в DEBUG

Можно использовать DEBUG для сохранения программы на диске в двух случаях.

1. Для получения с диска существующей программы, внесения в нее изменений и последующего сохранения.
2. Для создания при помощи DEBUG маленькой программы в машинных кодах, которую нужно сохранить.

За детальным объяснением обращайтесь к описанию команды W (write -записать) На данном этапе может оказаться полезной команда H, складывающая и вычитающая шестнадцатеричные числа. Максимальная длина чисел – 4 шестнадцатеричных разряда. Например, введите команду H 3443 2A2B. Команда выведет сначала сумму (5E6E), а затем – разность (0A18).

DEBUG также можно использовать для ввода программ на языке Ассемблера.

Для сохранения программы на диске сначала задается имя файла:

```
-N my_file.com
```

Затем в регистр CX необходимо поместить размер программы в байтах. Он будет равен разности конечного и начального смещений. Теперь остается только осуществить запись на диск командой W и в результате увидеть записанное количество байтов. В итоге мы получаем программу, готовую к исполнению.

Выход осуществляется командой q. Пример:

```
-A
```

```
0B3B:0100 mov ax,1234
0B3B:0103 mov ah, 4c
0B3B:0105 int 21
0B3B:0107
-u CS:100, 106
0B3B:0100 B83412 MOV AX,1234
0B3B:0103 B44C MOV AH,4C
0B3B:0105 CD21 INT 21
-r cx
CX 0000
:7
-r
AX=0000 BX=0000 CX=0007 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B3B ES=0B3B SS=0B3B CS=0B3B IP=0100 NV UP EI PL NZ NA PO NC
0B3B:0100 B83412 MOV AX,1234
-N my.com
-W
Запись 00007 байт
-q
```

Для выполнения этой лабораторной работы понадобится несколько команд ассемблера:

- MOV AH,<шестнадцатиричное число> - запись в регистр AH числа 02 для указания системной функции - вывод символа на экран;

- MOV DL,<шестнадцатиричное число> - запись в регистр DL кода символа;

- INT 21 - основное прерывание DOS (процедура), реализующее много различных функций; номер функции записывается предварительно в регистр AH; для распечатки символа на экране - в регистре AH функция 02, при этом в DL записывают предварительно код символа.

- INT 20 - прерывание DOS, осуществляющее выход из программы (из .COM-программы).

Пример:

Вывести символ "*" на экран.

```
mov AH,02 ; системная функция 02 - вывод символа на экран
mov DL,2A ; ASCII-код звездочки
int 21h ; прерывание для вывода "*"
int 20h ; выход из программы
```

Лабораторные работы

Синтаксис ассемблера FASM для работы с числами.

По умолчанию, число в программе воспринимается ассемблером как десятичное. Чтобы обозначить двоичное число, необходимо к нему в конце добавить символ *'b'*.

Восьмеричное число обозначается аналогично с помощью символа *'o'*. Для записи шестнадцатеричного числа FASM поддерживает 3 формы записи:

- перед числом записываются символы *'0x'* (как в C/C++);
- перед числом записывается символ *'$'* (как в Pascal);
- после числа записывается символ *'h'*. Если шестнадцатеричное число начинается с буквы, необходимо добавить в начале ноль (иначе непонятно, число это или имя метки).

Этот синтаксис используется как при объявлении данных, так и в командах. Вот примеры записи чисел во всех четырёх системах:

```
mov ax,537          ;Десятичная система

mov bl,11010001b    ;Двоичная система

mov ch,57o          ;Восьмеричная система

mov dl,$C2          ;\
mov si,0x013A       ; \
mov ah,18h          ; / Шестнадцатеричная система
mov al,0FFh         ;/

mov al,FFh          ;Ошибка!
```

Для записи отрицательного числа в программе на ассемблере используется символ *'-'*, например:

```
x db -5
```

Кстати, это работает и с числами в других системах счисления, и даже с символами

```
y db -25h
z db -77o
k db -101b
s db -'a'
```

Со знаковыми и беззнаковыми числами нужно быть внимательным, потому что только вы знаете, какие числа используются в вашей программе! Процессору абсолютно безразлично, какие данные он обрабатывает, поэтому невнимательность может привести к ошибке. Один и тот же байт может интерпретироваться по-разному, в зависимости от того со знаком число или без. Например, числу со знаком -5 соответствует число без знака 251:

Диапазоны значений чисел со знаком и без.

При программировании на ассемблере (как, впрочем, и на многих других языках) необходимо учитывать ещё один важный момент. А именно – ограничение диапазона представления чисел. Например, если размер беззнаковой переменной равен 1 байт, то она может принимать всего 256 различных значений. Это означает, что мы не сможем представить с её помощью число, больше 255 (11111111_2). Для такой же переменной со знаком максимальным значением будет 127 (01111111_2), а минимальным -128 (10000000_2). Аналогично определяется диапазон для 2- и 4-байтных переменных.

Кстати, так как процессор Intel 8086 был 16-битным и обрабатывал за одну команду 16-бит, то 16-битная переменная называется *слово (word)*, а 32-битная – *двойное слово (double*

word, dword). Эти названия сохранились в ассемблере даже для 32-битных процессоров (и в WIN32 API, например). И от них же происходят названия директив *dw (Define Word)* и *dd (Define Dword)*. Ну а *db* – это *Define Byte*.

Для наглядности вот табличка диапазонов чисел:

Размер переменной	**Число без знака**		**Число со знаком**	
	min	**max**	**min**	**max**
байт	00000000	11111111	10000000	01111111
	0	255	-128	127
слово	00000000 00000000	11111111 11111111	10000000 00000000	01111111 11111111
	0	65 535	-32 768	32 767
двойное слово	0000…0000	1111…1111	1000…0000	0111…1111
	0	4 294 967 295	-2 147 483 648	2 147 483 647
и т.д.	…	…	…	…

Если результат какой-то операции выйдет за пределы диапазона представления чисел, то случится *переполнение* и результат будет некорректным. (Например, при сложении двух положительных чисел, можно получить отрицательное число!) Поэтому нужно быть внимательным при программировании и предусмотреть обработку таких ситуаций, если они могут возникнуть.

Сложение

Для сложения двух чисел предназначена команда ADD. Она работает как с числами со знаком, так и с числами без знака (это особенность дополнительного кода).

Операнды должны иметь одинаковый размер (нельзя складывать 16- и 8-битное значение). Результат помещается на место первого операнда. В общем, эти правила справедливы для большинства команд.

После выполнения команды изменяются флаги, по которым можно определить характеристики результата:

- Флаг *CF* устанавливается, если при сложении произошёл перенос из старшего разряда. Для беззнаковых чисел это будет означать, что произошло переполнение и результат получился некорректным.
- Флаг *OF* обозначает переполнение для чисел со знаком.
- Флаг *SF* равен знаковому биту результата (естественно, для чисел со знаком, а для беззнаковых он равен старшему биту и особо смысла не имеет).
- Флаг *ZF* устанавливается, если результат равен 0.
- Флаг *PF* – признак чётности, равен 1, если результат содержит нечётное число единиц.

Примеры:

```
add ax,5          ;AX = AX + 5
add dx,cx         ;DX = DX + CX
```

```
add dx,cl        ;Ошибка: разный размер операндов.
```

Вычитание

Вычитание выполняется с помощью команды SUB. Результат также помещается на место первого операнда и опять же выставляются флаги. Единственная разница в том, что происходит вычитание, а не сложение.

На самом деле вычитание в процессоре реализовано с помощью сложения. Процессор меняет знак второго операнда на противоположный, а затем складывает два числа. Если вам необходимо в программе поменять знак числа на противоположный, можно использовать команду NEG. У этой команды всего один операнд.

Примеры:

```
sub si,dx        ;SI = SI - DX
neg ax           ;AX = -AX
```

Инкремент и декремент

Очень часто в программах используется операция прибавления или вычитания единицы. Прибавление единицы называется инкрементом, а вычитание – декрементом. Для этих операций существуют специальные команды процессора: INC и DEC. Обратите внимание, что эти команды не изменяют значение флага *CF*.

Пример программы

Требуется вычислить значение формулы: *e=a-(b+c-1)+(-d)*. Все числа являются 8-битными целыми со знаком. Объявим их после кода и придумаем какие-нибудь значения.

```
1  use16                ;Генерировать 16-битный код
2  org 100h             ;Программа начинается с адреса 100h
3
4      mov al,[a]      ;Загружаем значение a в AL
5      mov ah,[b]      ;Загружаем значение b в AH
6      add ah,[c]       ;AH = AH + c = b+c
7      dec ah           ;AH = AH - 1 = b+c-1
8      sub al,ah        ;AL = AL - AH = a-(b+c-1)
9      mov cl,[d]       ;CL = d
10     neg cl           ;CL = -CL = -d
11     add al,cl        ;AL = AL + CL = a-(b+c-1)+(-d)
12     mov [e],al       ;Сохраняем результат в e
13
14     mov ax,4C00h     ;\
15     int 21h          ;/ Завершение программы
16 ;---------------------------------------------------------------------------
17 a db 2
18 b db 3
19 c db 5
20 d db -8
21 e db ?
```

Квадратные скобки означают, что операнд находится по адресу, указанному внутри этих скобок. Так как вместо имени переменной FASM подставляет её адрес, то такая запись позволяет прочитать или записать значение переменной.

Запустив программу в Turbo Debugger, можно посмотреть её выполнение по шагам. Значения переменных можно увидеть в окне дампа памяти. Для этого нужно кликнуть правой кнопкой в этом окне и выбрать в меню пункт *Goto…*. Переменные начинаются в памяти с адреса 011Fh (этот адрес в первой команде).

Упражнение

Напишите программу для вычисления формулы *k=m+1-(n-1-r)*. Все числа 16-битные целые со знаком. Запустите в отладчике и проверьте правильность вычисления.

Сложение и вычитание с переносом

В системе команд процессоров x86 имеются специальные команды сложения и вычитания с учётом флага переноса (CF). Для сложения с учётом переноса предназначена команда ADC, а для вычитания – SBB. В общем, эти команды работают почти также, как ADD и SUB, единственное отличие в том, что к младшему разряду первого операнда прибавляется или вычитается дополнительно значение флага CF.
Зачем нужны такие команды? Они позволяют выполнять сложение и вычитание многобайтных целых чисел, длина которых больше, чем разрядность регистров процессора (в нашем случае 16 бит). Принцип программирования таких операций очень прост – длинные числа складываются (вычитаются) по частям. Младшие разряды складываются(вычитаются) с помощью обычных команд ADD и SUB, а затем последовательно складываются(вычитаются) более старшие части с помощью команд ADC и SBB. Так как эти команды учитывают перенос из старшего разряда, то мы можем быть уверены, что ни один бит не потеряется Этот способ похож на сложение(вычитание) десятичных чисел в столбик.

Аналогичная ситуация возникает с вычитанием чисел по частям. Допустим, требуется вычислить значение формулы k=i+j-n+1, где переменные k, i, j и n являются 32-битными целыми числами без знака. Складывать и вычитать такие числа придётся в два этапа: сначала вычисления будут производиться с младшими словами операндов, а затем со старшими с учётом переноса.
Для прибавления единицы в данном примере нельзя использовать команду INC, так как она не влияет на флаг CF и мы можем получить ошибку в результате!

```
use16               ;Генерировать 16-битный код
org 100h            ;Программа начинается с адреса 100h
   mov ax,word[i]    ;Загружаем младшую часть i в AX
   mov bx,word[i+2]  ;Загружаем старшую часть i в BX
   add ax,word[j]    ;Складываем младшие части i и j
   adc bx,word[j+2]  ;Складываем старшие части i и j
   sub ax,word[n]
   sbb bx,word[n+2]  ;BX:AX = i+j-n
   add ax,1          ;Команда INC здесь не подходит!
   adc bx,0          ;BX:AX = i+j-n+1
   mov word[k],ax    ;\
   mov word[k+2],bx  ;/ Сохраняем результат в k
    mov ax,4C00h     ;\
   int 21h           ;/ Завершение программы
;---------------------------------------------------------------
i dd 120000
j dd  80500
n dd  2300
k dd     ?
```

Запись word[i] означает, что мы переопределяем размер переменной (она объявлена как DWORD) и обращаемся к младшему слову. Старшее слово расположено в памяти после младшего, поэтому к адресу переменной надо прибавить 2, и соответствующая запись будет иметь вид word[i+2].
Посмотреть работу программы можно в отладчике.
Обратите внимание, как хранятся переменные в памяти. В процессорах Intel младший байт всегда хранится по младшему адресу, поэтому получается, что в окне дампа значения надо читать справа налево. В регистрах же числа записываются в нормальном виде. Сравните, как выглядит одно и то же значение k в памяти и в регистрах (старшая часть находится в BX, а младшая – в AX).
Одно из преимуществ ассемблера в том, что на нём можно реализовать работу с собственными форматами чисел, например с очень длинными целыми. А в языках высокого уровня выбор всегда ограничен компилятором. Следующая программа складывает два 7-байтных

```
use16               ;Генерировать 16-битный код
org 100h             ;Программа начинается с адреса 100h

    mov ax,word[x]
    add ax,word[y]
    mov word[z],ax
    mov ax,word[x+2]
    adc ax,word[y+2]
    mov word[z+2],ax
    mov ax,word[x+4]
    adc ax,word[y+4]
    mov word[z+4],ax
    mov al,byte[x+6]
    adc al,byte[y+6]
    mov byte[z+6],al
     mov ax,4C00h      ;\
    int 21h           ;/ Завершение программы
;---------------------------------------------------------------
x dd 0xF1111111
  dw 0xF111
  db 0x11
y dd 0x22222222
  dw 0x2222
  db 0x22
z rb 7
```

Обращение к старшему байту записывается как byte[x+6].
Упражнение
Напишите программу для вычисления формулы d=b-1+a-c. Все числа – 3-х байтные целые без знака. Проверьте работу программы в отладчике.

Умножение и деление

Умножение и деление выполняются по-разному для чисел со знаком и без, поэтому в системе команд процессора x86 есть отдельные команды умножения и деления для чисел со знаком и для чисел без знака.

Умножение чисел без знака

Для умножения чисел без знака предназначена команда MUL. У этой команды только один операнд – второй множитель, который должен находиться в регистре или в памяти. Местоположение первого множителя и результата задаётся неявно и зависит от размера операнда:

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

Отличие умножения от сложения и вычитания в том, что разрядность результата получается в 2 раза больше, чем разрядность сомножителей. Также и в десятичной системе – например, умножая двухзначное число на двухзначное, мы можем получить в результате максимум четырёхзначное. Запись «DX:AX» означает, что старшее слово результата будет находиться в DX, а младшее – в AX. Примеры:

```
mul bl      ;AX = AL * BL
mul ax      ;DX:AX = AX * AX
```

Если старшая часть результата равна нулю, то флаги *CF* и *OF* будут иметь нулевое значение. В этом случае старшую часть результата можно отбросить. Это свойство можно использовать в программе, если результат должен быть такого же размера, как множители.

Умножение чисел со знаком

Для умножения чисел со знаком предназначена команда IMUL. Эта команда имеет три формы, различающиеся количеством операндов:

- *С одним операндом* – форма, аналогичная команде MUL. В качестве операнда указывается множитель. Местоположение другого множителя и результата определяется по таблице.
- *С двумя операндами* – указываются два множителя. Результат записывается на место первого множителя. Старшая часть результата в этом случае игнорируется. Кстати, эта форма команды не работает с операндами размером 1 байт.
- *С тремя операндами* – указывается положение результата, первого и второго множителя. Второй множитель должен быть непосредственным значением. Результат имеет такой же размер, как первый множитель, старшая часть результата игнорируется. Это форма тоже не работает с однобайтными множителями.

Примеры:

```
imul cl             ;AX = AL * CL
imul si             ;DX:AX = AX * SI
imul bx,ax          ;BX = BX * AX
imul cx,-5          ;CX = CX * (-5)
imul dx,bx,134h     ;DX = BX * 134h
```

CF = *OF* = 0, если произведение помещается в младшей половине результата, иначе *CF* = *OF* = 1. Для второй и третьей формы команды *CF* = *OF* = 1 означает, что произошло переполнение.

Деление чисел без знака

Деление целых двоичных чисел – это всегда деление с остатком! По аналогии с умножением, размер делителя, частного и остатка должен быть в 2 раза меньше размера делимого. Деление чисел без знака осуществляется с помощью команды DIV. У этой команды один операнд – делитель, который должен находиться в регистре или в памяти. Местоположение делимого, частного и остатка задаётся неявно и зависит от размера операнда:

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

При выполнении команды DIV может возникнуть *прерывание* (о прерываниях я подробно расскажу потом, пока старайтесь избегать таких случаев):

- если делитель равен нулю;
- если частное не помещается в отведённую под него разрядную сетку (например, если при делении слова на байт частное больше 255).

Примеры:

```
div cl    ;AL = AX / CL, остаток в AH
div di    ;AX = DX:AX / DI, остаток в DX
```

Деление чисел со знаком

Для деления чисел со знаком предназначена команда IDIV. Единственным операндом является делитель. Местоположение делимого и частного определяется также, как для команды DIV. Эта команда тоже генерирует прерывание при делении на ноль или слишком большом частном.

Пример программы

Допустим, в программе требуется вычислять координату какого-то движущегося объекта по формуле:

$x = x_0 + v_0 t + at^2/2$

Все числа в правой части – 8-битные целые без знака, а *x* – 16-битное целое и тоже без знака. Здесь нужно внимательно следить за размерами операндов.

```
use16                ;Генерировать 16-битный код
org 100h             ;Программа начинается с адреса 100h

    mov al,[v0]      ;AL = v0
    mov cl,[t]       ;CL = t
```

```
    mul cl              ;AX = AL*CL = v0*t
    mov bx,ax           ;BX = AX = v0*t

    mov al,[a]          ;AL = a
    mul cl              ;AX = AL*CL = a*t
    mov ch,0            ;Преобразуем t в слово в регистре CX
    mul cx              ;DX:AX = AX*CX = a*(t^2)
    mov cl,2            ;CL = 2 = CX, так как CH = 0
    div cx              ;AX = DX:AX/2 = a*(t^2)/2

    add ax,bx           ;AX = AX+BX = v0*t + a*(t^2)/2
    add al,[x0]         ;\
    adc ah,ch           ;/ AX = AX+x0 = x0 + v0*t + a*(t^2)/2

    mov [x],ax          ;Сохраняем результат в x

    mov ax,4C00h        ;\
    int 21h             ;/ Завершение программы
;-------------------------------------------------------------------------------
x0  db 188
v0  db 7
a   db 3
t   db 25
x   dw ?
```

В 7-й строке промежуточный результат сохраняется в bx. В 11-й строке происходит преобразование байта в слово, путём добавления нулевой старшей части. Такой метод подходит для чисел без знака, но приведёт к ошибке для чисел со знаком (в случае отрицательного числа). Прибавление x_0 происходит в два этапа (строки 17 и 18) с учётом переноса, так как мы складываем слово и байт.

Упражнение

Напишите программу для вычисления формулы $z = (x \cdot y) / (x + y)$. Все числа 16-битные целые со знаком.

32-битное умножение

Команда MUL в 16-битном режиме позволяет умножать максимум 16-битные значения, поэтому для умножения 32-битных чисел необходим специальный алгоритм.
Умножать большие числа приходится по частям, а затем складывать эти промежуточные результаты. Допустим, нам надо умножить два 32-битных числа a и b. В результате должно получиться 64-битное число c. Обозначим как a1 – младшее слово a, a2 - старшее слово a, b1 – младшее слово b, b2 – старшее слово b. Тогда получается:

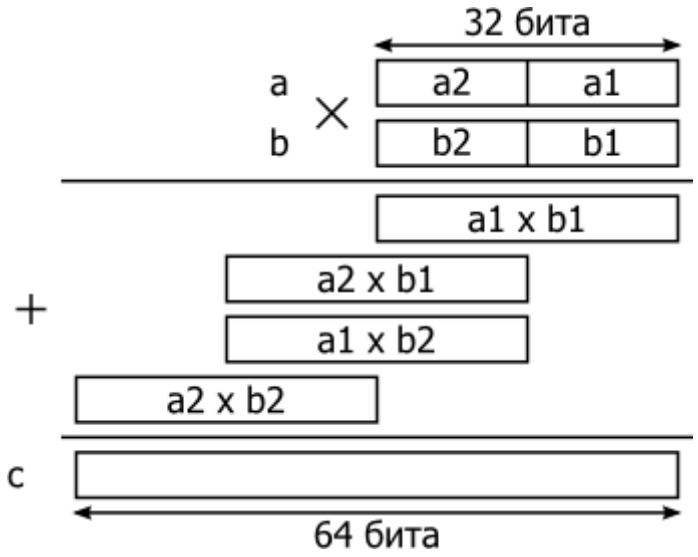
a = (a2<<16) + a1

b = (b2<<16) + b1

Символы <<16 обозначают сдвиг влево на 16 бит. По сути это то же самое, что умножить на 65536.

c = a x b = ((a2<<16) + a1) x ((b2<<16) + b1) =
= (a1 x b1) + ((a2 x b1)<<16) + ((a1 x b2)<<16) + ((a2 x b2)<<32)

Алгоритм очень напоминает способ умножения в столбик. Графически можно изобразить следующим образом:



Таким образом, в программе будет 4 этапа умножения и 3 сложения промежуточных результатов (обязательно с учётом переноса):

```
1  use16                    ;Генерировать 16-битный код
2  org 100h                 ;Программа начинается с адреса 100h
3
4  ;Полный результат умножения будет находиться в регистрах BX:CX:SI:DI
5
6      mov ax,word[a]       ;AX = младшее слово a
7      mul word[b]          ;Умножение на младшее слово b
8      mov di,ax            ;DI = младшая часть результата
9      mov si,dx            ;SI = старшая часть результата
10     sub cx,cx            ;CX = 0
11     sub bx,bx            ;BX = 0
12
13     mov ax,word[a+2]     ;AX = старшее слово a
14     mul word[b]          ;Умножение на младшее слово b
15     add si,ax            ;\ Прибавление промежуточного результата
16     adc cx,dx            ;/ с учётом переноса
17
18     mov ax,word[a]       ;AX = младшее слово a
19     mul word[b+2]        ;Умножение на старшее слово b
20     add si,ax            ;\
21     adc cx,dx            ; > Прибавление промежуточного результата
22     adc bx,bx            ;/  с учётом переноса
23
24     mov ax,word[a+2]     ;AX = старшее слово a
25     mul word[b+2]        ;Умножение на старшее слово b
26     add cx,ax            ;\ Прибавление промежуточного результата
27     adc bx,dx            ;/ с учётом переноса
28
29     mov word[c],di       ;\
30     mov word[c+2],si     ; \
31     mov word[c+4],cx     ; / Сохранение результата в переменной c
32     mov word[c+6],bx     ;/
33
34     mov ax,4C00h         ;\
35     int 21h              ;/ Завершение программы
```

```
36 ;--------------------------------------------------------------------------------------------
37 align 4
38 a   dd $FFFFFFFF
39 b   dd $FFFFFFFF
40 c   dq ?
```

Подобным образом можно выполнять умножение чисел любой разрядности, причём необязательно одинаковой. ***Проверьте умножение следующих чисел, 32 х 64 бит, 16 х 32 бит и т.д.***

Приложение 2

ШКАЛА ОЦЕНИВАНИЯ ПИСЬМЕННОГО ОПРОСА
(промежуточный контроль – «ЗНАТЬ»)

- Отметкой **(9-10 баллов)** оценивается ответ, который показывает прочные знания основных понятий криптографии, алгоритмов шифрования данных, математических основ применяемых алгоритмов.
- Отметкой **(7-8 баллов)** оценивается ответ, обнаруживающий прочные знания основных понятий криптографии, алгоритмов шифрования данных, математических основ применяемых алгоритмов. Однако допускается одна - две неточности в ответе.
- Отметкой **(4-6 баллов)** оценивается ответ, свидетельствующий в основном о знании основных понятий криптографии, алгоритмов шифрования данных, математических основ применяемых алгоритмов. Допускается несколько ошибок в содержании ответа.
- Отметкой **(2-3 балла)** оценивается ответ, обнаруживающий незнание основных понятий криптографии, алгоритмов шифрования данных, математических основ применяемых алгоритмов. Отмечается отсутствие логичности и последовательности в ответе. Допускаются серьезные ошибки в содержании ответа.
- Отметкой **(0 -1 балл)** оценивается ответ, при котором студент демонстрирует непонимание проблемы или нет ответа.

ШКАЛА ОЦЕНИВАНИЯ ПРАКТИЧЕСКИХ ЗАДАНИЙ
(промежуточный контроль – «УМЕТЬ и ВЛАДЕТЬ»)

- Отметкой **(17-20 баллов)** оценивается ответ, при котором студент правильно решает индивидуальные задачи по предложенному алгоритму шифрования, демонстрирует способность грамотно формализовать задачу из индивидуального задания.
- Демонстрирует полное понимание проблемы. Все требования, предъявляемые к заданию, выполнены.
- Отметкой **(11-16 баллов)** оценивается ответ, при котором студент в основном правильно решает индивидуальные задачи по предложенному алгоритму шифрования, демонстрирует способность грамотно формализовать задачу из индивидуального задания. Демонстрирует значительное понимание проблемы. Большинство требований, предъявляемых к заданию, выполнены.
- Отметкой **(4-10 баллов)** оценивается ответ, при котором студент в основном не правильно решает индивидуальные задачи по предложенному алгоритму шифрования, демонстрирует способность грамотно формализовать задачу из индивидуального задания. Демонстрирует частичное или небольшое понимание проблемы. Многие требования, предъявляемые к заданию, не выполнены.
- Отметкой **(0 -3 балла)** оценивается ответ, при котором студент демонстрирует непонимание проблемы или нет ответа и даже не было попытки решить задачу

ШКАЛА ОЦЕНИВАНИЯ РЕФЕРАТА

№	Наименование показателя	Отметка (в %)
1	Во введении четко сформулирован тезис, соответствующий теме реферата, выполнена задача заинтересовать читателя	85 - 100
2	Деление текста на введение, основную часть и заключение	
3	В основной части логично, связно и полно доказывается выдвинутый тезис	
4	Заключение содержит выводы, логично вытекающие из содержания основной части	
5	Все требования, предъявляемые к заданию, выполнены	
6	При защите реферата демонстрирует полное понимание проблемы и для выражения своих мыслей не пользуется упрощенно-примитивным языком.	
1	Во введении четко сформулирован тезис, соответствующий теме реферата, в известной мере выполнена задача заинтересовать читателя	75 – 84
2	В основной части логично, связно, но не достаточно полно доказывается выдвинутый тезис	
3	Заключение содержит выводы, логично вытекающее из содержания основной части	
4	При защите реферата демонстрирует понимание проблемы и для выражения своих мыслей не пользуется упрощенно-примитивным языком.	
1	Во введении тезис сформулирован не четко и не вполне соответствует теме реферата	60 - 74
2	В основной части выдвинутый тезис доказывается недостаточно логично (убедительно) и последовательно	
3	Заключенные выводы не полностью соответствуют содержанию основной части	
4	При защите реферата демонстрирует не полное понимание проблемы и язык работы в целом не соответствует требуемому уровню	
1	Во введении тезис отсутствует или не соответствует теме реферата	40 - 59
2	Деление текста на введение, основную часть и заключение отсутствует	
3	В основной части нет логичного последовательного раскрытия темы	
4	Выводы не вытекают из основной части	
5	При защите реферата демонстрирует полное непонимание проблемы и язык работы можно оценить, как «примитивный».	
1	Работа написана не по теме	менее 58

Раздел 4. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций

УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ (МОДУЛЯ)			
6.1. Рекомендуемая литература			
6.1.1. Основная литература			
	Авторы,	Заглавие	Издательство, год
Л1.1	Галисеев Г.В.	Ассемблер IBM PC: Самоучитель	Издательский дом "Вильямс" 2004. -304
Л1.2	Таненбаум Э.	Архитектура компьютера: учебное	СПб.: Питер 2013
6.1.2. Дополнительная литература			
	Авторы,	Заглавие	Издательство, год
Л2.1	Голубь Н.Г.	Искусство программирования на Ассемблере.: Лекции и	СПб.: ООО "ДиаСофтЮП" 2002
Л2.2	Таненбаум Э.	Архитектура компьютера: учебное	СПб.: Питер 2013
Л2.3	Майко Г.В.	Ассемблер для IBM PC: научно-	М.: Бизнес 1977
6.1.3. Методические разработки			
	Авторы,	Заглавие	Издательство, год
Л3.1	Касымалиева	Информатика: Учебно-методическое	Бишкек: Изд-во КРСУ 2016
Л3.2	Грошев А.С.	Информатика: лабораторный практикум: учебное пособие	Архангельск: Арханг. гос. техн. ун-т 2012
6.2. Перечень ресурсов информационно-телекоммуникационной сети "Интернет"			
Э1	Образовательный математический сайт Exponenta.ru....		old.exponenta.ru›soft/matlab/matlab_book.asp
Э2	Информационно-справочный портал		http://ru.wikipedia.org/wiki/
Э3	Публичная электронная библиотека		Public- library.narod.ru.
Э4	Российский общеобразовательный портал.		library.ru.

Раздел 5. Методические указания для обучающегося по освоению дисциплины. Математические методы решения прикладных задач.

Текущий контроль осуществляется в течение семестра и включает в себя проверку подготовки студента к выполнению
практической работы (т.е. знание теории по теме практической работы), проверку завершения практической работы и
оформления отчёта.

Рекомендации по организации самостоятельной работы студента

1. Советы по планированию и организации времени, необходимого для изучения дисциплины. Рекомендуется следующим
образом организовать время, необходимое для изучения дисциплины:

Изучение конспекта лекции в тот же день, после лекции – 10-15 минут.
Изучение конспекта лекции за день перед следующей лекцией – 10-15 минут.
Изучение теоретического материала по учебнику и конспекту – 1 час в неделю.
Подготовка к практическому занятию – 2 час.
Всего в неделю – 3 часа 30 минут.

2. Описание последовательности действий студента

Для понимания материала и качественного его усвоения рекомендуется такая последовательность действий:
2.1. После прослушивания лекции и окончания учебных занятий, при подготовке к занятиям следующего дня, нужно
сначала просмотреть и обдумать текст лекции, прослушанной сегодня (10-15 минут).
2.2. При подготовке к лекции следующего дня, нужно просмотреть текст предыдущей лекции, подумать о том, какая может
быть тема следующей лекции (10-15 минут).
2.3. В течение недели выбрать время (1-час) для работы с рекомендуемой литературой в библиотеке.
2.4. При подготовке к практическим занятиям следующего дня, необходимо сначала прочитать основные понятия и подходы
по теме домашнего задания. При выполнении упражнения или задачи нужно сначала понять, что требуется в задаче, какой
теоретический материал нужно использовать, наметить план решения задачи.
3. Рекомендации по использованию материалов учебно-методического комплекса. Рекомендуется использовать
методические указания по курсу , текст лекций преподавателя.

4. Рекомендации по работе с литературой. Теоретический материал курса становится более понятным, когда дополнительно
к прослушиванию лекции и изучению конспекта, изучаются и книги. Легче освоить курс, придерживаясь одного учебника и
конспекта. Рекомендуется, кроме «заучивания» материала, добиться состояния понимания изучаемой темы дисциплины. С
этой целью рекомендуется после изучения очередного параграфа выполнить несколько простых упражнений на данную
тему. Кроме того, очень полезно мысленно задать себе следующие вопросы (и попробовать ответить на них): о чем этот
параграф?, какие новые понятия введены, каков их смысл?, что даст это на практике?.

5. Советы по подготовке к рубежному и промежуточному контролям. Дополнительно к изучению конспектов лекции
необходимо пользоваться учебником. Кроме «заучивания» материала, очень важно добиться состояния понимания
изучаемых тем дисциплины. С этой целью рекомендуется после изучения очередного параграфа выполнить несколько
упражнений на данную тему. Кроме того, очень полезно мысленно задать себе следующие вопросы (и попробовать ответить
на них): о чем этот параграф?, какие новые понятия введены, каков их смысл?, что даст это на практике?.
При подготовке к промежуточному контролю нужно изучить теорию: определения всех понятий и подходы к оцениванию
до состояния понимания материала и самостоятельно решить несколько типовых задач из каждой темы. При решении задач
всегда необходимо уметь качественно интерпретировать итог решения.

6. Указания по организации работы с контрольно-измерительными материалами, по выполнению домашних заданий. При
выполнении домашних заданий необходимо сначала прочитать основные понятия и подходы по теме задания. При
выполнении упражнения или задачи нужно сначала понять, что требуется в задаче, какой теоретический материал нужно

использовать, наметить план решения задачи, а затем приступить к расчетам и сделать качественный вывод.
Рубежный контроль осуществляется в течение семестра в виде защиты практических работ, выполнения контрольных
работ, тестирования и контрольных вопросов.
Методические указания по выполнению практических работ представлены в электронной папке преподавателя (локальная
сеть кафедры Информационных и вычислительных технологий КРСУ). Выполнение практических работ завершается
оформлением отчета, в котором приводится теория по теме работы, результаты работы программы и графики.
Рекомендации по работе с тестами по дисциплине.
Совокупность тестов , представленная в разделе 5.3 может использоваться по частям для текущего контроля освоения
предмета по разделам и темам и для итогового контроля знаний по дисциплине в целом.
Предполагается «загрузить» тесты в специализированную ПС или использовать их в форме бланочного контроля. Они
могут использоваться студентами для самоконтроля при подготовке к итоговому зачёту по дисциплине.
Вопросы по контрольным работам представлены в разделе РПД Фонд оценочных средств.
Ответы на контрольные работы оформляются в письменном виде.
Список литературы должен быть составлен в полном соответствие с действующим стандартом (правилами).
Для допуска к зачету студент в течении семестра должен сдать все практические работы.