УТВЕРЖДЕНО

решением Учёного совета факультета математики, информационных и авиационных технологий

от «21» июня 2019 г., протокол № 5/19

Председатель _____/ М.А. Волков

«21» июня 2019 г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Дисциплина	Программирование на Ассемблере
Факультет	Факультет математики, информационных и авиационных технологий
Кафедра	Телекоммуникационные технологии и сети
Курс	4

Направление (специальность) 09.03.02 - "Информационные системы и технологии"

код направления (специальности), полное наименование

Направленность (профиль/специализация) Разработка информационных систем

полное наименование

Форма обучения очная, заочная

очная, заочная, очно-заочная

Дата введения в учебный процесс УлГУ: «1» сентября 2019 г.

Программа актуализирована на заседании кафедры: протокол №_____ от _____ 20_____г.
Программа актуализирована на заседании кафедры: протокол №_____ от _____ 20_____г.
Программа актуализирована на заседании кафедры: протокол №_____ от _____ 20_____г.

Сведения о разработчиках:

ФИО	Кафедра	Должность, ученая степень, звание
Липатова Светлана Валерьевна	Телекоммуникационных технологий и сетей	доцент, к.т.н., доцент

СОГЛАСОВАНО	СОГЛАСОВАНО
Заведующий кафедрой телекоммуникационных технологий и сетей, реализующей дисциплину	Заведующий выпускающей кафедрой телекоммуникационных технологий и сетей
(_____/ Смагин А.А ____/ *Подпись* *ФИО* «21» июня 2019 г.	(_____/ Смагин А.А ____/ *Подпись* *ФИО* «21» июня 2019 г.

1. ЦЕЛИ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ:

Цели освоения дисциплины: формирование общекультурных и профессиональных компетенций, необходимых для реализации информационно-аналитической и научно-исследовательской деятельности

Задачи освоения дисциплины: приобретение в рамках освоения предусмотренного курсом занятий следующих знаний, умений и навыков, характеризующих определённый уровень сформированности компетенций (см. подробнее п.3):

- изучение машинно-зависимых языков программирования (ассемблеров), основы построения и архитектуры ЭВМ, основы современных языков ассемблера.
- получение знаний о принципах построения языка ассемблера, ассемблерах разного типа, интегрированных сред разработки, поддерживающие работу на Ассемблере.

2. МЕСТО ДИСЦИПЛИНЫ В СТРУКТУРЕ ОПОП:

Дисциплина «Программирование на Ассемблере» относится к числу дисциплин блока Б1.В.ДВ., предназначенного для студентов второго курса, обучающихся по направлению 09.03.02 - "Информационные системы и технологии".

Для успешного изучения дисциплины необходимы знания и умения, приобретенные в результате освоения курсов «Информатика и программирование»; «Дискретная математика и математическая логика», и полностью или частично сформированные компетенции ОПК-1, УК-1, а именно:

- **знать:** основные понятия, утверждения, а так же методы исследования, методику построения различных дискретных структур, новейшие достижения дискретной математики, основные принципы программирования;
- **уметь:** применять методы дискретной математики на практике, работать в средах программирования;
- **владеть:** методологией и навыками решения научных и практических задач, писать программы на языках высокого уровня.

Основные положения дисциплины используются в дальнейшем при изучении таких дисциплин как: «Преддипломная практика»; «Выполнение и защита выпускной квалификационной работы».

3. ПЕРЕЧЕНЬ ПЛАНИРУЕМЫХ РЕЗУЛЬТАТОВ ОБУЧЕНИЯ ПО ДИСЦИПЛИНЕ (МОДУЛЮ), СООТНЕСЕННЫХ С ПЛАНИРУЕМЫМИ РЕЗУЛЬТАТАМИ ОСВОЕНИЯ ОСНОВНОЙ ПРОФЕССИОНАЛЬНОЙ ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ

Код и наименование реализуемой компетенции	Перечень планируемых результатов обучения по дисциплине соотнесённых с индикаторами достижения компетенций
ОПК-2 Способен использовать современные информационные технологии и программные средства, в том числе отечественного производства, при решении задач профессиональной деятельности	**Знать:** основы построения и архитектуры ЭВМ, **Уметь:** разрабатывать программы на машинно-зависимых языках программирования; **Владеть:** методами разработки алгоритмов программ на ассемблере, современными интегрированными средами разработки программного обеспечения

ПК-3 Способен использовать математические методы обработки, анализа и синтеза результатов исследований	**Знать:** элементы архитектуры ЭВМ **Уметь:** строить алгоритмы **Владеть:** операторами языка ассемблер
УК-2 Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений	**Знать:** возможности, недостатки и достоинства машинно-зависимых языков **Уметь:** писать программы на машинно-зависимых языках **Владеть:** средой разработки программного обеспечения

4. ОБЩАЯ ТРУДОЕМКОСТЬ ДИСЦИПЛИНЫ

4.1. Объем дисциплины в зачётных единицах (всего): 6

4.2. Объем дисциплины по видам учебной работы (в часах)

Форма обучения очная

Вид учебной работы	Количество часов (форма обучения очная)	
	Всего по плану	В т.ч. по семестрам
		8
1	2	3
Контактная работа обучающихся с преподавателем в соответствии с УП	56	56
Аудиторные занятия:	54	54
лекции	18	18
Семинары и практические занятия	18	18
Лабораторные работы, практикумы	18	18
Самостоятельная работа	54	54
Форма текущего контроля знаний и контроля самостоятельной работы	тестирование, контрольная работа (решение задач)	тестирование, контрольная работа (решение задач)
Курсовая работа	-	-
Виды промежуточной аттестации (экзамен, зачёт)	зачёт	зачёт
Всего часов по дисциплине	108	108

Форма обучения заочная

Вид учебной работы	Количество часов (форма обучения заочная)	
	Всего по плану	В т.ч. по курсам
1	2	3

Контактная работа обучающихся с преподавателем в соответствии с УП	26	26
Аудиторные занятия:	26	26
лекции	8	8
Семинары и практические занятия	10	10
Лабораторные работы, практикумы	8	8
Самостоятельная работа	78	78
Форма текущего контроля знаний и контроля самостоятельной работы	тестирование, контрольная работа (решение задач)	тестирование, контрольная работа (решение задач)
Курсовая работа	-	-
Виды промежуточной аттестации (экзамен, зачет)	зачёт	зачёт
Всего часов по дисциплине	108	108

4.3. Содержание дисциплины (модуля.) Распределение часов по темам и видам учебной работы:

Форма обучения очная

Название разделов и тем	Всего	Виды учебных занятий					Форма текущего контроля знаний
		Аудиторные занятия			Занятия в интерактивной форме	Самостоятельная работа	
		Лекции	Практические занятия, семинары	Лабораторные работы, практикумы			
1	2	3	4	5	6	7	8
1. Общие принципы организации ЭВМ.	14	2	2	-	-	10	Текущий контроль (проверка решения)
2. Основные элементы программирования на Ассемблере.	24	4	4	6	6	10	Текущий контроль (проверка решения) Проверка лабораторных работ
3. Сложные типы данных в Ассемблере: массивы, строки. Организация работы с одномерными	24	4	4	6	6	10	Текущий контроль (проверка решения) Проверка лабораторных работ

массивами и со строками.							
4.Сложные типы данных в Ассемблере: двумерные массивы, структуры и записи. Способы адресации операндов для работы со сложными типами данных.	24	4	4	6	6	10	Текущий контроль (проверка решения) Проверка лабораторных работ
5. Процедуры и макросы в Ассемблере, назначение и использование.	22	4	4	-	-	14	Текущий контроль (проверка решения)
Итого	108	18	18	18	18*	54	-

Форма обучения заочная

Название разделов и тем	**Всего**	**Виды учебных занятий**					**Форма текущего контроля знаний**
		Аудиторные занятия			**Занятия в интерактивной форме**	**Самостоятельная работа**	
		Лекции	**Практические занятия, семинары**	**Лабораторные работы, практикумы**			
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1. Общие принципы организации ЭВМ.	18	1	2			15	Текущий контроль (проверка решения)
2. Основные элементы программирования на Ассемблере.	20	1	2	2	2	15	Текущий контроль (проверка решения) Проверка лабораторных работ
3. Сложные типы данных в Ассемблере: массивы, строки. Организация работы с	22	2	2	3	3	15	Текущий контроль (проверка решения) Проверка лабораторных работ

одномерными массивами и со строками.							
4.Сложные типы данных в Ассемблере: двумерные массивы, структуры и записи. Способы адресации операндов для работы со сложными типами данных.	22	2	2	3	3	15	Текущий контроль (проверка решения) Проверка лабораторных работ
5. Процедуры и макросы в Ассемблере, назначение и использование.	22	2	2	-	-	18	Текущий контроль (проверка решения)
Итого	108	8	10	8	-	78	4

5. СОДЕРЖАНИЕ ДИСЦИЛИНЫ (МОДУЛЯ)

Тема 1. Общие принципы организации ЭВМ.

Области применения языков программирования низкого уровня. Поколения ПК IBM PC. Основные факторы, влияющие на рост производительности ВС. Архитектурные особенности современных ПК. Базовая архитектура ПК IBM PC , процессор с точки зрения программиста, регистры общего назначения, регистр флагов. Представление данных и команд, форматы команд, способы адресации операндов. Организация памяти, режимы работы процессора. Организация памяти в реальном режиме работы, сегментные регистры, понятие исполняемого и физического адреса.

Тема 2. Основные элементы программирования на Ассемблере.

Структура программы на Ассемблере, модели памяти, команды, директивы и комментарии. Алфавит, слова, константы, выражения, переменные. Стандартные директивы сегментации и упрощенные, (точечные), организация COM-файлов. Директивы определения данных и памяти. Команды пересылки безусловной и условной, команды загрузки адреса. Сегмент стека, организация работы со стеком, команды для работы со стеком, команды прерывания. Команды двоичной арифметики. Программирование линейных, разветвляющихся и циклических алгоритмов. Этапы решения задач на Ассемблере в среде операционной системы MS-DOS. Трансляция программ на Ассемблере, компоновка программ, отладка программ

Тема 3. Сложные типы данных в Ассемблере: массивы, строки. Организация работы с одномерными массивами и со строками.

Массивы, выделение памяти, работа с одномерными массивами. Команды для

работы со строками, организация работы со строками переменной длины. Адресация операндов для работы с массивами и со строками.

Тема 4.Сложные типы данных в Ассемблере: двумерные массивы, структуры и записи. Способы адресации операндов для работы со сложными типами данных.

Организация работы с двумерными массивами. Структуры в Ассемблере, их описание и использование. Команды побитовой обработки данных. Записи в Ассемблере, их описание и использование, команды для работы с записями. Способы адресации для работы со структурами и записями.

Тема 5. Процедуры и макросы в Ассемблере, назначение и использование.

Работа с подпрограммами в Ассемблере, способы передачи параметров. Передача параметров через стек, локальные параметры в процедуре, организация рекурсивных процедур. Макросредства в языке Ассемблер, блоки повторений и макросы. Директивы условной генерации. Примеры использования макросредств.

6. ТЕМЫ ПРАКТИЧЕСКИХ И СЕМИНАРСКИХ ЗАНЯТИЙ

Тема 1. Общие принципы организации ЭВМ.

1) Каковы принципы организации ЭВМ?
2) Какие есть уровни организация вычислительных процессов?
3) Что означает магистральность?
4) Опишите структур процессора.
5) Чем отличаются архитектуры CISC и RISC?
6) Что такое программа?
7) Что такое программное обеспечение?
8) Что такое прикладное программирование?
9) Назовите основные этапы подготовки программы.
10) Назовите основные виды директив.
11) Что такое сегментные регистры?
12) Назовите основные виды сегментных регистров.
13) Что такое регистры общего назначения?
14) Какие основные регистры общего назначения Вы можете назвать?
15) Что такое регистровые указатели?
16) Какие регистровые указатели Вы знаете?
17) Что такое индексные регистры?
18) Назовите основные индексные регистры?
19) Что такое регистр командного указателя?
20) Что такое флаговый регистр?
21) Для чего нужен флаговый регистр?

Тема 2. Основные элементы программирования на Ассемблере.

1) Что такое сегмент в языке программирования Ассемблер?
2) Что такое расширение набора команд в языке программирования Ассемблер?
3) Что такое способы адресации в языке программирования Ассемблер?
4) Что такое директивы в языке программирования Ассемблер?
5) Основные этапы умножения в языке программирования Ассемблер.
6) Основные этапы деления в языке программирования Ассемблер.

7) Расскажите о прямом табличном доступе.
8) Какие группы регистров выделяются в микропроцессоре и каковы особенности их использования?
9) Какую функцию в микропроцессоре выполняет регистр флагов?
10) Как используется регистр команд IP?
11) Какие шаги необходимо выполнить для получения из программы на языке ассемблера исполняемого модуля?
12) Перечислите арифметические команды с подробной характеристикой каждой.
13) Перечислите логические команды с подробной характеристикой каждой.
14) Расшифруйте параметры 10 функции 21го прерывания.
15) Приведите два способа ввода числа с клавиатуры.
16) Напишите алгоритм преобразования строки в число.
17) Напишите алгоритм преобразования числа в строку.
18) Опишите работу команды сравнения cmp.
19) Перечислите переходы по одному флагу.
20) Перечислите переходы по нескольким флагам (условные).
21) Приведите пример использования команды сравнения и команд условного перехода.

Тема 3. Сложные типы данных в Ассемблере: массивы, строки. Организация работы с одномерными массивами и со строками.

1) Перечислите все цепочные команды с параметрами.
2) Опишите подробно работу команды LODS.
3) Опишите подробно работу команды STOS.
4) Опишите подробно работу команды SCAS.
5) Опишите подробно работу команды MOVS.
6) Опишите подробно работу команды CMPS.
7) Опишите подробно работу префикса повторения REP.
8) Напишите аналог префикса повторения REP.
9) Перечислите и подробно опишите режимы адресации.

Тема 4.Сложные типы данных в Ассемблере: двумерные массивы, структуры и записи. Способы адресации операндов для работы со сложными типами данных.

1) Опишите способ задания одномерного и многомерного массивов.
2) Опишите способы обращения к элементам массива.
3) Опишите работу команды LOOP с примером.
4) Опишите синтаксис создания процедуры с примером.
5) Опишите команды работы со стеком.
6) Дан двумерный массив, найти среднее арифметическое элементов главной диагонали.
7) Дан двумерный массив, записать в качестве элементов побочной диагонали суммы по строкам.
8) Дан двумерный массив, отсортировать его по возрастанию.
9) Дан двумерный массив, поменять местами главную и побочную диагонали.

Тема 5. Процедуры и макросы в Ассемблере, назначение и использование.

1) Нарисуйте программную архитектуру математического сопроцессора.
2) Перечислите регистры математического сопроцессора, подробно опишите работу регистра тегов.

3) Опишите подробно регистр состояния математического сопроцессора.
4) Опишите подробно регистр управления математического сопроцессора.
5) Перечислите основные типы арифметических команд математического сопроцессора.
6) Перечислите основные арифметические команды математического сопроцессора.
7) Перечислите основные трансцендентные команды математического сопроцессора.
8) Перечислите и подробно опишите особые случаи математического сопроцессора.
9) Опишите способы обработки особых случаев математического сопроцессора.
10) Опишите команды перехода в графический и текстовый режимы.
11) Что такое байт-атрибут?
12) Перечислите доступные режимы работы в графическом режиме.
13) Перечислите доступные палитры цветов.
14) Опишите функции 10го прерывания для работы в графическом режиме.
15) Напишите программу, выводящую горизонтальную линию на экран.
16) Напишите программу, выводящую несколько линий, образующих звезду, на экран.
17) Напишите программу, выводящую вертикальную линию на экран.
18) Напишите программу, заполняющую экран через строку точками разных цветов.
19) Опишите работу функции создания файла с примером.
20) Опишите работу функции открытия существующего файла с примером.
21) Опишите работу функции открытия и создания файла с примером.
22) Опишите работу функции чтения файла с примером.
23) Опишите работу функции записи в файл с примером.
24) Опишите работу функции переименования файла с примером.
25) Опишите работу функции удаления файла с примером.
26) Опишите работу функции закрытия файла с примером.
27) Опишите работу функции поиска файла с примером.

7. ЛАБОРАТОРНЫЕ РАБОТЫ, ПРАКТИКУМЫ

Тема 2. Основные элементы программирования на Ассемблере.

Задание: Загрузите и выполните предназначенную для системы DOS (дисковая операционная система) программу вывода строки на экран.

1.1 *Порядок выполнения задания*

Шаг 1. Основы программирования для DOS

Изучите программы вывода на экран строки «Hello, World!» из примеров 1,2,3.

Теоретические сведения

1.1.1.1 Структура программы

Язык ассемблера – это машинно-ориентированный язык низкого уровня, в котором каждое высказывание соответствует одной инструкции процессора. Так что можно сказать, что ассемблер это удобная человеку запись процессорных команд. Чтобы перевести эту запись в понимаемые процессором коды, нужна особая программа, которая тоже называется ***ассемблером***.

Программа ассемблер – преобразует исходный текст в коды команд процессора. Текст программы на ассемблере содержит кроме инструкций еще и служебную информацию, которая помогает программе ассемблеру понять, что же от нее требуется.
Таким образом, программа на языке ассемблера состоит из операторов и директив. Операторы – это инструкции, управляющие работой процессора, а директивы указывают программе ассемблеру, каким образом следует объединять инструкции для создания модуля, который и станет работающей программой.
Инструкция процессора на языке ассемблера состоит не более чем из четырех полей и имеет следующий формат:
[[метка:]] мнемоника [[операнды]] [[;комментарии]]
В квадратных скобках находятся необязательные поля. Единственное обязательное поле – поле кода операции (мнемоника), определяющее инструкцию, которую должен выполнить микропроцессор. Поле операндов определяется кодом операции и содержит дополнительную информацию о команде. Каждому коду операции соответствует определенное число операндов. Метка служит для обозначения какого-то определенного места в памяти, т. е. содержит в символическом виде адрес, по которому храниться инструкция. Преобразование символических имен в действительные адреса осуществляется программой ассемблера. Часть строки исходного текста после символа «;» (если он не является элементом знаковой константы или строки знаков) считается комментарием и ассемблером игнорируется. Комментарии вносятся в программу как поясняющий текст и могут содержать любые знаки до ближайшего символа конца строки. Для создания комментариев, занимающих несколько строк, может быть использована директива COMMENT. Пример:
Метка Код операции Операнды ;Комментарий
MET: MOVE AX, BX ;Пересылка
Структура директивы аналогична структуре инструкции. Второе поле – код псевдооперации определяет смысловое содержание директивы. Как и у инструкции, у директивы есть операнды, причем их может быть один или несколько и они отделяются друг от друга запятыми. Допустимое число операндов в директиве определяется кодом псевдооперации. Пример:
ARRAY DB 0, 0, 0, 0, 0
END START
Директива может быть помечена символическим именем и содержать поле комментария. Символическое имя, стоящее в начале директивы распределения памяти, называется ***переменной***. В отличие от метки команды после символического имени директивы двоеточие не ставиться. Комментарий записывается также как и в случае команды и может занимать целую строку.

1.1.1.2 Организация программы

Программа на языке ассемблера состоит из программных модулей, содержащихся в различных файлах. Каждый модуль, в свою очередь, состоит из инструкций процессора или директив ассемблера и заканчивается директивой END. Метка, стоящая после кода псевдооперации END, определяет адрес, с которого должно начаться выполнение программы и называется ***точкой входа в программу***.
Каждый модуль также разбивается на отдельные части ***директивами сегментации***, определяющими начало и конец сегмента. Каждый сегмент начинается директивой начала сегмента – SEGMENT и заканчивается директивой конца сегмента – ENDS . В начале директив ставится имя сегмента.
При загрузке программ в оперативную память DOS инициализирует как минимум три сегментных регистра: CS, DS и SS. При этом совокупности байтов, представляющих

команды процессора (код программы), и данные помещаются из файла на диске в оперативную память, а адреса этих сегментов записываются в CS и DS соответственно. Сегмент стека либо выделяется в области, указанной в программе, либо совпадает (если он явно в программе не описан) с самым первым сегментом программы. Адрес сегмента стека помещается в регистр SS. Дополнительный сегментный регистр ES часто используется для обращения к полям данных, не входящим в программу, например, к видеопамяти или системным ячейкам. Однако при необходимости его можно настроить на один из сегментов программы.
Все сегменты могут использовать различные области памяти, а могут частично или полностью перекрываться (рис.1.4).
В дальнейшем любые обращения к ячейкам программы осуществляются путем указания сегмента, в котором находится интересующая нас ячейка, а так же номера того байта внутри сегмента, к которому мы хотим обратиться. Этот номер носит название ***относительного адреса*** или ***смещения***.
Адрес начала сегмента определяют по следующему правилу: нужно умножить содержимое сегментного регистра на 16, что эквивалентно сдвигу числа на четыре двоичных разряда влево.
При записи команд на языке ассемблера принято указывать адреса с помощью следующей конструкции:
<адрес сегмента>:<смещение> или
<сегментный регистр>:<адресное выражение>
Например, запись DS:DX означает, что адрес ячейки складывается из адреса начала сегмента, хранящегося в регистре DS, и смещения внутри сегмента, записанного в DX.
Программа ассемблер должна знать заранее, через какие сегментные регистры будут адресоваться ячейки программы и чтобы сообщить ей об этом используется директива ASSUME (« assume » – предположить).

1.1.1.3 Директива ASSUME

С помощью директивы ASSUME ассемблеру сообщается информация о соответствии между сегментными регистрами, и программными сегментами. Директива имеет следующий формат:
ASSUME <пара>[[, <пара>]]
ASSUME NOTHING
где <пара> - это <сегментный регистр> :<имя сегмента>
либо <сегментный регистр> :NOTHING
Например, директива
ASSUME ES:A, DS:B, CS:C
сообщает ассемблеру, что для сегментирования адресов из сегмента A выбирается регистр ES, для адресов из сегмента B – регистр DS, а для адресов из сегмента C – регистр CS.
Таким образом, директива ASSUME дает право не указывать в командах (по крайней мере, в большинстве из них) префиксы – опущенные префиксы будет самостоятельно восстанавливать ассемблер.
Если в директиве ASSUME указано несколько пар с одним и тем же сегментным регистром, то последняя из них «отменяет» предыдущие, т. к. каждому сегментному регистру, можно поставить в соответствие только один сегмент. В то же время на один и тот же сегмент могут указывать разные сегментные регистры. Если в директиве ASSUME в качестве второго элемента пары задано служебное слово NOTHING (ничего), например, ASSUME ES:NOTHING, то это означает, что с данного момента сегментный регистр не указывает ни на какой сегмент, что ассемблер не должен использовать этот регистр при трансляции команд.

В связи с тем, что директивой ASSUME автор программы лишь сообщает, что все имена из таких-то программных сегментов должны сегментироваться по таким-то сегментным регистрам (в начале выполнения программы в этих регистрах ничего нет), то выполнение программы необходимо начинать с команд, которые загружают в сегментные регистры адреса соответствующих сегментов памяти. При этом в директиве обязательно должно быть указано соответствие между регистром CS и данным сегментом кода, иначе при появлении первой же метки ассемблер зафиксирует ошибку.
После загрузки программы в память DOS инициирует оба сегментных регистра DS и ES значением адреса ***префикса программного сегмента*** PSP (Program Segment Prefix) – специальной области оперативной памяти размером 256 (100h) байт. PSP может использоваться в программе для определения имен файлов и параметров из командной строки, введенной при запуске программы на выполнение, объема доступной памяти, переменных окружения системы и т.д.
Таким образом, чтобы DS указывал на сегмент данных программы, а не на блок PSP, его инициализируют вручную. Пусть регистр DS необходимо установить на начало сегмента B. Для загрузки регистра выполняется присваивание вида DS:=B. Однако сделать это командой MOV DS,B нельзя, поскольку процессор устроен так, что он не может непосредственно передать имя сегмента в сегментный регистр, поэтому делаем это через посредника – регистр AX:

```
MOV AX,B
MOV DS,AX ;DS:=B
```

Аналогичным образом загружается и регистр ES.
Регистра CS загружать нет необходимости, так как к началу выполнения программы этот регистр уже будет указывать на начало сегмента кода. Такую загрузку выполняет операционная система, прежде чем передает управление программе.
Загрузить регистр SS можно двояко. Во-первых, его можно загрузить в самой программе так же, как DS. Во-вторых, такую загрузку можно поручить операционной системе, описав в программе сегмент стека с помощью ключевого слова STACK.

1.1.1.4 EXE- и COM-программы

DOS может загружать и выполнять программные файлы двух типов – COM и EXE. Ввиду сегментации адресного пространства процессора 8086 оба типа программ могут выполняться в любом месте памяти. Программы никогда не пишутся в предположении, что они будут загружаться с определенного адреса (за исключением некоторых самозагружающихся, защищенных от копирования программ).
EXE-программы содержат несколько программных сегментов, включая сегмент кода, данных и стека. EXE-файл загружается, начиная с адреса PSP:0100h. В процессе загрузки считывается информация EXE-заголовка в начале файла, при помощи которого загрузчик выполняет настройку ссылок на сегменты в загруженном модуле, чтобы учесть тот факт, что программа была загружена в произвольно выбранный сегмент. После настройки ссылок управление передается загрузочному модулю к адресу CS:IP, извлеченному из заголовка EXE.
COM-программы содержат единственный сегмент (или, во всяком случае, не содержат явных ссылок на другие сегменты). Образ COM-файла считывается с диска и помещается в память, начиная с PSP:0100h , в связи с этим COM -программа должна содержать в начале сегмента кода директиву позволяющую осуществить такую загрузку (ORG 100h).
COM-программы предпочтительнее EXE-программ, когда дело касается небольших ассемблерных утилит. Они быстрее загружаются, ибо не требуется перемещения сегментов, и занимают меньше места на диске, поскольку EXE-заголовок и сегмент стека отсутствуют в загрузочном модуле.

Ниже приведены примеры, иллюстрирующие основные особенности структуры EXE - и COM -программ, написанных на языке ассемблера:

Пример 1. Структура EXE -программы, использующей «классический» способ задания сегментов с помощью директивы SEGMENT

```
stak segment stack 'stack'      ;Начало сегмента стека
db 256 dup (?)                  ;Резервируем 256 байт для стека
stak ends                       ;Конец сегмента стека
data segment 'data'             ;Начало сегмента данных
Hello db 'Hello, World!$'       ;Строка для вывода
data ends                       ;Конец сегмента данных
code segment 'code'             ;Начало сегмента кода
assume CS:code,DS:data,SS:stak  ;Сегментный регистр CS будет указывать на сегмент команд,
                                ;регистр DS - на сегмент данных, SS – на стек
start:                          ;Точка входа в программу start
;Обязательная инициализация регистра DS в начале программы
mov AX,data                     ;Адрес сегмента данных сначала загрузим в AX,
mov DS,AX                       ;а затем перенесем из AX в DS
mov AH,09h                      ;Функция DOS 9h вывода на экран
mov DX,offset Hello             ;Адрес начала строки 'Hello, World!' записывается в регистр DX
int 21h                         ;Вызов функции DOS
mov AX,4C00h                    ;Функция 4Ch завершения программы с кодом возврата 0
int 21h                         ;Вызов функции DOS
code ends                       ;Конец сегмента кода
end start                       ;Конец текста программы с точкой входа
```

Замечание. При вводе исходного текста программы можно использовать как прописные, так и строчные буквы: имена stak и STAK будут для ассемблера одинаковыми.

В общем случае, взаимное расположение сегментов EXE -программы может быть любым, но чтобы сократить в командах число ссылок вперед и избежать проблем с префиксами для них, рекомендуется сегмент команд размещать в конце текста программы.

Сегмент стека в приведенной структуре описан с параметром STACK, поэтому в самой программе нет необходимости загружать сегментный регистр SS. Сегментный регистр CS тоже нет необходимости загружать, как уже отмечалось ранее. В связи с этим в начале программы загружается лишь регистр DS.

Вывод на экран строки текста и выход из программы осуществляются путем вызова стандартных процедур DOS, называемых ***прерываниями***. Прерывания под номером 21h (33 – в десятичной системе счисления) называются ***функциями DOS***, у них нет названий, а только номера. Номер прерывания и его параметры передаются в регистрах процессора, при этом номер должен находиться в регистре AH. Так, например, прерывание INT 21h, с помощью которого на экран выводится строка символов, управляется двумя параметрами: в регистре AH должно быть число 9, а в регистре DX – адрес строки символов, оканчивающейся знаком '$'.

Адрес строки Hello загружается в регистр DX с помощью оператора OFFSET (смещение):

OFFSET имя

Значением оператора является смещение указанного имени, отсчитанное от начала того сегмента, в котором оно описано. Имя может быть именем переменой, или меткой (именем команды). При трансляции ассемблер ставит в соответствие имени переменной

Hello, описанной в директиве DB, смещение первого байта строки Hello относительно начала сегмента DS. С помощью оператора OFFSET мы записываем это смещение в DX. Выход из программы осуществляется через функцию DOS с номером 4Ch. Эта функция предполагает, что в регистре AL находится код завершения программы, который она передаст DOS. Если программа завершилась успешно, код завершения должен быть равен 0, поэтому в примере 1 загружаем регистры AH и AL с помощь одной команды MOV ax,4C00h , после чего вызываем прерывание 21h.
«Классический» способ задания сегментов удобен для написания больших программ на ассемблере, состоящих из разнообразных модулей и содержащих множество сегментов. В повседневном программировании обычно используется ограниченный набор простых вариантов организации программы, известных как модели памяти.
Модель памяти задается директивой .MODEL, после установления которой, вступают в силу упрощенные директивы определения сегментов, объединяющие действия директив SEGMENT и ASSUME. Кроме того, сегменты, объявленные упрощенными директивами, не требуется закрывать директивой ENDS – они закрываются автоматически, как только ассемблер обнаруживает новую директиву определения сегмента или конец программы.
Пример 2. Структура EXE -программы, использующей модель памяти SMALL и упрощенные директивы определения сегментов

```
.model small              ;Модель памяти SMALL использует сегменты
                          ;размером не более 64Кб
.stack 100h               ;Сегмент стека размером 100h (256 байт)
.data                     ;Начало сегмента данных
Hello db 'Hello, World!$'
.code                     ;Начало сегмента кода
start:                    ;Точка входа в программу start
                          ;Предопределенная метка @data обозначает
                          ;адрес сегмента данных в момент запуска программы,
mov AX, @data             ;который сначала загрузим в AX,
mov DS,AX                 ;а затем перенесем из AX в DS
mov AH,09h
mov DX,offset Hello
int 21h
mov AX,4C00h
int 21h
end start
```

Принципиальной особенностью COM -программы является то, что в отличие от EXE -программы, которая содержит отдельные сегменты данных, стека и кода, COM -программ содержит лишь один основной сегмент (сегмент кода), в котором размещаются и код и данные и стек.
Пример 3. Структура COM -программы, использующей модель памяти TINY и упрощенные директивы определения сегментов

```
.model tiny               ;Модель памяти TINY, в которой код, данные и стек
                          ;размещаются в одном и том же сегменте размером до 64Кб
.code                     ;Начало сегмента кода
org 100h                  ;Устанавливает значение программного счетчика в 100h
                          ;Начало необходимое для COM-программы,
                          ;которая загружается в память с адреса PSP:100h
start:
mov AH,09h
```

```
mov DX,offset Hello
int 21h
mov AX,4C00h
int 21h
;===== Data =====
Hello db 'Hello, World!$'
end start
```

Шаг 2. Подготовка программы к выполнению

Создайте файл с программой из примера 1. Подготовьте программу к выполнению. Запустите программу и убедитесь, что она выводит на экран нужный текст и успешно завершается.

Теоретические сведения

Процесс подготовки и отладки программы на языке ассемблера включает этапы подготовки файла с исходным текстом, его трансляции и компоновки.

Подготовка исходного текста программы выполняется с помощью любого текстового редактора. Файл с исходным текстом должен иметь расширение .ASM.

Трансляция исходного текста программы состоит в преобразовании предложений исходного языка в коды машинных команд и выполняется с помощью транслятора с языка ассемблера, т.е. с помощью программы ассемблера. Существует несколько версий ассемблера – это, например, пакет MASM фирмы Microsoft, или пакет TASM корпорации Borland.

Рассмотрим работу с пакетом TASM (транслятор TASM.EXE , редактор связей TLINK.EXE, отладчик TD.EXE) более подробно.

Входной информацией для ассемблера (TASM.EXE) является исходный файл — текст программы на языке ассемблера в кодах ASCII. В результате работы ассемблера может получиться до 3-х выходных файлов:

1) объектный файл – представляет собой вариант исходной программы, записанный в машинных командах;

2) листинговый файл – является текстовым файлом в кодах ASCII, включающим как исходную информацию, так и результат работы программы ассемблера;

3) файл перекрестных ссылок – содержит информацию об использовании символов и меток в ассемблерной программе.

Если ассемблер во время ассемблирования обнаруживает ошибки, он записывает сообщения о них в листинговый файл. Кроме того, он выводит их на экран дисплея.

Имена файлов исходного (.ASM), объектного (.OBJ), листинга (.LST) и перекрестных ссылок (.XRF) указываются ассемблеру непосредственно в командной строке через запятую, например:

TASM test, Otest, Ltest, Ctest

Для создания только объектного файла достаточно указать одно имя исходного файла:

TASM test

В результате получим объектный файл test.obj, которому ассемблер присвоит то же имя, что и у исходного, но с расширением OBJ.

Ключ /L позволяет создавать наряду с объектным файлом листинг трансляции с тем же именем, что и у исходного файла:

TASM /L test

Из листинга трансляции легко определить размер отдельных составляющих программы и всей программы в целом.

Замечание. Как в тексте программы на языке ассемблера, так и при вводе с клавиатуры командных строк можно с равным успехом использовать и прописные и строчные буквы.

Программа, полученная в результате ассемблирования (объектный файл), еще не готова к выполнению. Ее необходимо обработать программой редактирования связей TLINK, которая может связать несколько различных объектных модулей в одну программу и на основе объектного модуля формирует исполняемый загрузочный модуль.
Входной информацией для программы TLINK являются имена объектных модулей. Если файлов больше одного, то их имена вводятся через разделитель «+». Модули связываются в том же порядке, в каком их имена передаются программе TLINK .
Если в командной строке не указано имя выходного исполняемого модуля, TLINK по умолчанию присваивает ему имя первого из объектных модулей с расширением .EXE. Вводя другое имя, можно изменять имя файла, но не расширение.
Редактор связей TLINK (компоновщик), и программа, выдающая объектный файл, управляются ключами – символами, стоящими непосредственно за косой чертой:
TLINK /x test
В данном случае компоновщиком управляет один ключ /x , который подавляет формирование карты загрузки (файла с листингом компоновки test.map), без которого вполне можно обойтись.
Как уже отмечалось, компоновщик создает загрузочный, готовый к исполнению модуль в формате .EXE. Запуск подготовленной программы осуществляется командой
test.exe
или просто
test
Для получения исполняемого файла в формате .COM из исходного файла testcom.asm , предназначенного для создания COM -программ, необходимо указать компоновщику TLINK ключ /t :
TLINK /x /t testcom
Графически процесс создания программы на языке ассемблера можно представить, как это показано на рис.1.6.

1.1.1.5 Создание командного файла

Автоматизируем выполнение однотипных операций трансляции и компоновки, создав файл make.bat , в котором содержатся команды программе ассемблеру. Текст командного файла может быть таким:

```
cls
tasm.exe %1.asm
tlink.exe /x %1.obj
%1
```

Приведенный текст составлен в предположении, что путь к программам пакета TASM указан в переменной PATH. Если это по каким-либо причинам не так, в командный файл следует включить полную спецификацию файлов ассемблера и компоновщика, например, если весь пакет находится на диске D : в каталоге TASM , указываем:
d:\tasm\tasm.exe %1.asm
Командный файл make.bat помещается в папку, где хранится исходный текст программы, или в одну из папок, указанных в переменной PATH.
После того, как командный файл создан, остается перейти в каталог с исходным файлом test.asm, набрать в командной строке:
make test
и нажать Enter.
Команда make test использует имя файла без расширения. Расширения .asm и .obj «приклеиваются» справа от имени test, когда выполняется командный файл, и в результате получаются команды:

cls
tasm.exe test.asm
tlink.exe /x test.obj
test

Шаг 3. Отладка программы

Выполните отладку программы из примера 1 с помощью итеративного отладчика из пакета TASM.

Теоретические сведения

1.1.1.6 Отладчик пакета TASM

Отладку готовой программы наиболее удобно выполнять с помощью какого-либо итеративного отладчика, который позволяет исполнять программу по шагам или с точками останова, выводить на экран содержимое регистров и областей памяти, модифицировать (в известных пределах) загруженную в память программу, принудительно изменять содержимое регистров и выполнять другие действия, позволяющие в наглядной и удобной форме контролировать выполнение программы. При трансляции и компоновке программы с помощью пакета TASM следует использовать Turbo Debugger (файл TD.EXE).

Для отладки полученного exe-файла необходимо выполнить команду DOS:

td имя_файла.exe

Произойдет запуск отладчика и загрузка EXE -файла в него. В процессе работы отладчик использует также исходный файл, поэтому перед отладкой оба файла должны находиться в рабочем каталоге.

На сообщение об отсутствии дополнительной отладочной информации следует нажать ОК. Окно отладчика представлено на рис.1.7.

После вызова отладчика и загрузки в память отлаживаемой программы, ее первое предложение помечается знаком . По мере выполнения программных предложений значок будет перемещаться по тексту программы, всегда указывая на очередное (еще не выполненное) предложение.

Для выполнения программы по шагам предназначены клавиши F7 и F8. Разница между ними заключается в обработке команды CALL. Нажатие F7 приводит к переходу на первую команду вызываемой подпрограммы, а нажатие F8 – к переходу к следующей за CALL команды вызывающей подпрограммы. Комбинация клавиш Ctrl+F2 позволяет заново начать выполнение программы с ее первой команды. Клавиша F4 позволяет выполнить программу до текущей строки.

В соответствующих окнах также можно просматривать содержимое регистров и стека, переходить между окнами можно с помощью мыши или клавиши Tab. Комбинация клавиш Alt+F10 открывает для любого активного окна внутреннее меню с дополнительным возможностями.

Для того чтобы, находясь в отладчике, увидеть результат работы программы, надо использовать комбинацию клавиш Alt+F5. Возврат в кадр отладчика осуществляется нажатием любой клавиши.

Тема 3. Сложные типы данных в Ассемблере: массивы, строки. Организация работы с одномерными массивами и со строками.

Задание . Создайте массив из 10 байт, в который запишите числа от 1 до 10. Выведите все числа этого массива на экран в одном столбце, выравнивая их по правому краю:

1.2 *Порядок выполнения*

Шаг 1. Структура и алгоритм работы программы

Изучите предлагаемую структуру программы и алгоритм ее работы.

```
.model small
.stack 100h ;Сегмент стека
.186 ;Разрешение трансляции команд процессора 80186
.data ;Сегмент данных
;Массив simple, содержащий числа от 1 до 10
;. . .
;Строка символов result, определяющая формат вывода чисел на
экран
;. . .
;Строка nl для перевода курсора вниз и возврата к левому краю
экрана
;. . .
.code ;Сегмент кода
start: ;Точка входа в программу
mov ax, @data ;Инициализация сегментного
mov ds, ax ;регистра DS
;Задаем предусловие для организации
;цикла ввода чисел в массив simple
;a.Число шагов в цикле.
;. . .
;b.Начальный элемент массива simple.
;. . .
;c.Индекс начального элемента массива simple.
;. . .
vvod: ;Цикл, в котором записываем числа от 1 до 10
;в последовательные ячейки массива simple
loop vvod
;Задаем предусловие для организации
; цикла вывода чисел массива simple на экран:
;a.Число шагов в цикле.
;. . .
;b.Индекс начального элемента массива simple.
;. . .
vyvod: ;Цикл вывода элементов массива на экран,
;в котором вызывается процедура byte_asc
;преобразования числа в строку
loop vyvod
mov ax,4C00h ;Завершение программы
int 21h
byte_asc proc ;Процедура преобразования числа в строку
pusha ;Располагает в стеке все регистры общего назначения
;в следующем порядке: AX,CX,DX,BX,SP,BP,SI,DI.
;В паре с командой popa, которая считывает
;эти же регистры из стека в обратном порядке
;это позволяет писать подпрограммы,
;которые не должны изменять значения регистров
```

```
;по окончании своей работы.
;Команды pusha/popa поддерживаются начиная с процессоров 80186.
;Поскольку необходимо выводить числа в столбец,
;выравнивая по правому краю,
;будем сохранять отдельные цифры числа не в стеке,
;а в специальной строке result, справа налево.
popa
ret ;Возврат в точку вызова
byte_asc endp ;Конец процедуры
end start ;Конец программы
```

1.2.1.1 Алгоритм цикла ввода чисел в массив

Предусловие:

a. В регистре CX находится число шагов в цикле, равное 10.
b. В регистре BL находится начальный элемент массива, равный 1.
c. В регистре SI находится индекс начального элемента массива, равный 0.

Тело цикла:

1. Заносим содержимое BL в очередной байт массива simple[SI].
2. Увеличиваем SI на 1, чтобы перейти к следующему байту массива.
3. Увеличиваем BL на 1, чтобы получить следующее число для занесения в массив.

1.2.1.2 Алгоритм цикла вывода

Предусловие:

a. В регистре CX находится число шагов в цикле, равное 10.
b. В регистре SI находится индекс начального элемента массива, равный 0.

Тело цикла:

1. Заносим очередной байт массива simple[SI] в регистр AL.
2. Заносим основание системы счисления, равное 10, в региср BL.
3. Вызываем процедуру byte_asc.
4. Организуем вывод результата на экран:
a. помещаем номер 9 функции DOS в регистр AH;
b. заносим в DX адрес result (см. л.р.№1);
c. вызываем прерывание 21h;
d. заносим в DX адрес nl (см. л.р.№1);
e. вызываем прерывание 21h.
5. Увеличиваем SI на 1, чтобы перейти к следующему байту массива.

1.2.1.3 Алгоритм процедуры преобразования числа в строку

Предусловие: выводимое число находится в регистре AL, основание системы счисления - в регистре BL.

Начало.

1. Заносим длину строки результата (=2) в регистр SI.
2. Задаем позицию в строке результата для очередной цифры числа (уменьшаем индекс в регистре SI на 1).
3. Заносим 0 в регистр AH, т.к. при делении на байт в качестве делимого выступает регистр AX. (В AL до вызова процедуры уже помещено выводимое число).
4. Выполняем деление AX на BL: неполное частное от деления помещается в AL, а остаток - в AH.
5. Получаем из цифры, соотвествующей остатку от деления, код цифры в таблице ASCII (добавляем к цифре (AH) код нуля).
6. Заносим код цифры в result[SI].
7. Проверка условия: если неполное частное (AL) не равно 0, то переход на пункт 2.

Конец.

Шаг 2. Написание сегмента данных

Объявите в сегменте данных:

1. массив с именем simple из 10 байтов с неопределенным значением;
2. строку символов с именем result, содержащую 2 пробела и символ '**$**', завершающий строку;
3. последовательность управляющих символов с именем nl, состоящую из символа **перевода строки** с кодом `0Ah`, символа **возврата каретки** с кодом `0Dh` и символа конца строки '**$**'

Теоретические сведения

Сегмент данных предназначен для хранения полей данных программы таких, как глобальные переменные. Место под переменные отводится директивами определения данных.

1.2.1.4 Директивы определения данных

Директивы определения данных служат для задания размеров, содержимого и местоположения полей данных, используемых в программе на языке ассемблера.

Директивы определения скалярных данных, представляющих собой единичное значение или набор единичных значений, приведены в таблице 4.1.

Таблица 4.1. Директивы определения скалярных данных.

Формат	Функция
`[имя] DB значение,...`	определение байтов
`[имя] DW значение,...`	определение слов
`[имя] DD значение,...`	определение двойных слов
`[имя] DQ значение,...`	определение квадрослов
`[имя] DT значение,...`	определение 10 байтов

Директива DB обеспечивает распределение и инициализацию 1 байта памяти для каждого из указанных значений. В качестве значения может кодироваться целое число, текстовая строка, оператор DUP (см. ниже), абсолютное выражение или знак «?». Знак «?» обозначает неопределенное значение. Значения, если их несколько, должны разделяться запятыми. Если директива имеет имя, создается переменная типа BYTE с соответствующим данному значению указателя позиции смещением.

Если в одной директиве определения памяти заданы несколько значений, им распределяются последовательные байты памяти. В этом случае, имя, указанное в начале директивы, именует только первый из этих байтов, остальные остаются безымянными.

Текстовые строки вводятся в программу с помощью директивы DB и заключаются в апострофы или в кавычки. Символы строки хранятся в памяти в порядке их следования, т.е. 1-й символ имеет самый младший адрес, последний - самый старший.

Во всех директивах определения памяти в качестве одного из значений может быть задан оператор DUP. Он имеет следующий формат:

```
счетчик DUP (значение,...)
```

Указанный в скобках список значений повторяется многократно в соответствии со значением счетчика. Каждое значение в скобках может быть любым выражением, имеющим значением целое число, символьную константу или другой оператор DUP (допускается до 17 уровней вложенности операторов DUP). Значения, если их несколько, должны разделяться запятыми.

Оператор DUP может использоваться не только при определении памяти, но и в других директивах.

Синтаксис директив DW, DD, DQ и DT идентичен синтаксису директивы DB.

Примеры директив определения скалярных данных:

```
integer1 DB 16
string1 DB 'abCDf$'
string2 DB 2 dup(' '), '$'
empty1 DB ?
contan2 DW 4*3
string3 DD 'ab'
high4 DQ 18446744073709551615
high5 DT 120892581961462917470617 5d
db5 DB 5 DUP(5 DUP(5 DUP(10)))
db6 DB 5 DUP(10)
dw6 DW 10 DUP(?)
nl DB 0Dh, 0Ah, '$' ;вывод nl на экран переводит курсор
;на начальную позицию следующей строки
```

Шаг 3. Заполнение массива числами от 1 до 10

Следуя алгоритму цикла ввода чисел в массив, заполните массив simple числами от 1 до 10.

Теоретические сведения

1.2.1.5 Режимы адресации

В зависимости от спецификаций и местоположения источников образования полного (абсолютного) адреса в языке ассемблера различают следующие способы адресации операндов:

- регистровая;
- прямая;
- непосредственная;
- косвенная;
- базовая;
- индексная;
- базово-индексная.

Регистровая адресация подразумевает использование в качестве операнда регистра процессора, например:

```
PUSH DS
MOV BP,SP
```

При прямой адресации один операнд представляет собой адрес памяти, второй – регистр:

```
MOV Data,AX
```

Непосредственная адресация применяется, когда операнд, длинной в байт или слово находится в ассемблерной команде:

```
MOV AH,4Ch
```

При использовании косвенной адресации абсолютный адрес формируется исходя из сегментного адреса в одном из сегментных регистров и смещения в регистрах BX , BP , SI или DI :

```
MOV AX,ES:[SI] ;База – в ES, смещение – в SI
MOV AL,[BX] ;База – в DS, смещение – в BX
;если в команде не указан сегментный регистр,
;по умолчанию используется DS
MOV AX,[BP] ;База – в SS, смещение – в BP
;при использовании регистра BP процессор
;воспринимает смещение как смещение в сегменте стека
```

В случае применения базовой адресации исполнительный адрес является суммой значения смещения и содержимого регистра BP или BX , например:

```
MOV AX,[BP+6] ;База – SS, смещение – BP+6
MOV DX,8[BX] ;База – DS, смещение – BX+8
```

При индексной адресации исполнительный адрес определяется как сумма значения указанного смещения и содержимого регистра SI или DI так же, как и при базовой адресации, например:

```
MOV DX,[SI+5] ;База – DS, смещение – SI+5
```

Базово-индексная адресация подразумевает использование для вычисления исполнительного адреса суммы содержимого базового регистра и индексного регистра, а также смещения, находящегося в операторе, например:

```
MOV BX,[BP][SI] ;База – SS, смещение – BP+SI
MOV ES:[BX+DI],AX ;База – ES, смещение – BX+DI
MOV AX,[BP+6+DI] ;База – SS, смещение - BP+6+DI
```

1.2.1.6 Базовая и индексная адресации со смещением

Исполнительный адрес определяется суммой содержимого регистра BX, BP, SI или DI и указанного в команде числа, которое называют смещением.

Пример:

```
MAS db 1,2,5,3,7,9,8,3,4 ;Массив чисел
mov BX,2 ;BX = индекс элемента в массиве
mov dl, MAS[BX] ;В DL заносится второй байт массива
```

В этом примере исполнительный адрес элемента массива под номером 2 (***нумерация в массиве ведется с нуля***) вычисляется как сумма содержимого BX =2 и адреса начала массива MAS .

В результате в регистр DL будет загружено число 5.

Предполагается, что базовый адрес сегмента, в который входит массив mas, загружен в DS. Таким образом, имя массива MAS ассемблер считает относительным адресом начала массива в сегменте данных.

Инструкцию `mov DL,MAS[BX]` можно переписать как `mov DL,[MAS+BX]` , где адрес в квадратных скобках равен сумме адреса, связанного с именем массива и относительного адреса (***относительно начала массива***), хранящегося в регистре BX . Очевидно, инструкция `mov DL,[MAS]` запишет в DL нулевой элемент массива. Это же можно проделать инструкцией `mov DL,MAS`.

Адресация в массиве с помощью регистров SI и DI осуществляется аналогично и называется ***индексной адресацией со смещением***:

```
mov DL,MAS[SI].
```

При использовании регистра BP следует помнить, что в качестве сегментного регистра по умолчанию подразумевается SS.

Шаг 4. Вывод массива

Реализуйте вывод формируемых из элементов массива simple строк на экран, следуя алгоритму цикла вывода.

Шаг 5. Процедура преобразования числа в строку

Реализуйте процедуру byte_asc, следуя п.1-п.7. алгоритма преобразования числа в строку.

Тема 4.Сложные типы данных в Ассемблере: двумерные массивы, структуры и записи. Способы адресации операндов для работы со сложными типами данных.

Задание. Используя функции прерывания INT 10h, напишите EXE - программу, которая выводит на экран таблицу ASCII в окне размером 16x16 и координатами левого верхнего угла 4:24.

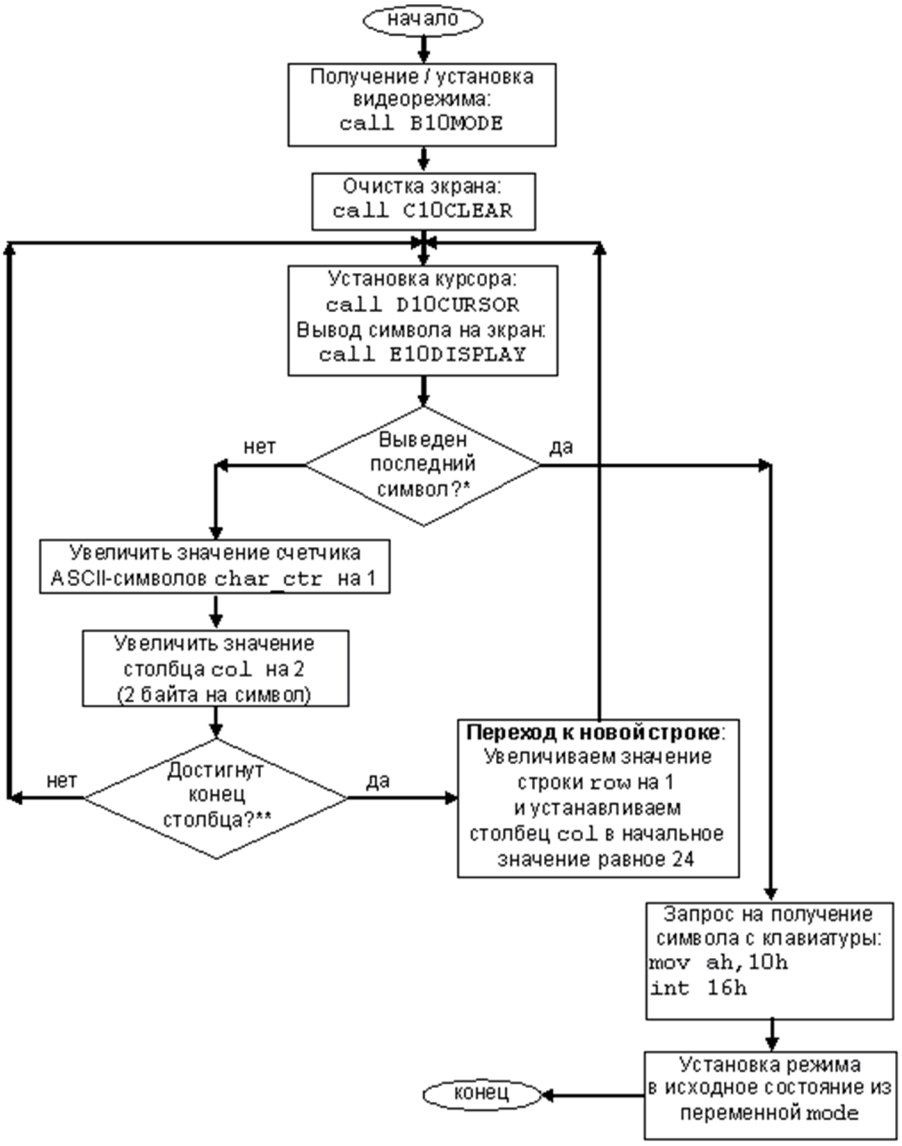
1.3 *Порядок выполнения*

Структура и алгоритм работы программы вывода символов ASCII средствами BIOS

Изучите предлагаемую структуру программы и алгоритм ее работы.

```
.model small
.stack 100h
.186
.data
;Определение переменных:
;row – строка, в которой находится курсор, начальное значение
равно 4
;col – столбец, в котором находится курсор, начальное значение
равно 24
;mode – номер режима, начальное значение не определено (равно ?)
;char_ctr – счетчик ASCII-символов, начальное значение равно 0
.code
start:
mov ax,@data
mov ds,ax
;Код главной программы
;. . .
mov ax,4C00h        ;Завершение программы
int 21h
;Определение подпрограмм
B10MODE proc        ;Получение/установка видеорежима
;Код подпрограммы
;. . .
ret
B10MODE endp
C10CLEAR proc       ;Очистка экрана
;Код подпрограммы
;. . .
ret
C10CLEAR endp
D10CURSOR proc      ;Установка курсора
;Код подпрограммы
;. . .
ret
D10CURSOR endp
E10DISPLAY proc     ;Вывод символа на экран
;Код подпрограммы
;. . .
ret
E10DISPLAY endp
end start           ;Конец программы
```

1.3.1.1 Блок – схема главной программы



*Для проверки условия «Выведен последний символ?» сравните значение счетчика ASCII-символов char_ctr с 255.
**Для проверки условия «Достигнут конец столбца?» сравните значение столбца col с правой границей окна равной 56 (=24+16*2).

1.3.2 Алгоритмы подпрограмм

1.3.2.1 B10MODE – получение и сохранение текущего режима, установка нового режима

Начало.
1. Получаем текущий режим в регистре AL: в регистр AH заносим значение 0Fh после чего вызывается прерывание 10h.
2. Сохраняем номер текущего режима: заносим в mode содержимое регистра AL.
3. Переводим экран в текстовый режим с номером 03: в регистр AH помещается значение 00h, в регистр AL – номер режима 03, после чего вызывается прерывание 10h.
4. Выполняем возврат из подпрограммы в точку вызова.
Конец.

1.3.2.2 C10CLEAR – очистка экрана, выделение окна, задание атрибутов

Начало.
1. Помещаем в стек все регистры общего назначения.
2. Прокручиваем вверх весь экран целиком (см. функция 06h).
3. Создаем окно с 16 строками и 16 столбцами: в регистр AH заносим значение 06h, в регистр AL – число строк 10h, в регистр BH – значение атрибута - зеленый фон, белые символы, в регистр CX – координаты левого верхнего угла 04:24, в регистр DX – координаты правого нижнего угла 19:54, после чего вызывается прерывание 10h.
4. Загружаем из стека все регистры общего назначения.
5. Выполняем возврат из подпрограммы в точку вызова.
Конец.

1.3.2.3 D10CURSOR - позиционирование курсора в заданные строку и столбец

Начало.
1. Помещаем в стек все регистры общего назначения.
2. Устанавливаем курсор: в регистр AH заносим значение 02h, в регистр BH – номер страницы 0, в регистр DH – значение строки row , в регистр DL – значение столбца col, после чего вызывается прерывание 10h.
3. Загружаем из стека все регистры общего назначения.
4. Выполняем возврат из подпрограммы в точку вызова.
Конец.

1.3.2.4 E10DISPLAY - вывод ASCII-символа

Начало.
1. Помещаем в стек все регистры общего назначения.
2. Выводим символ с установленным атрибутом на экран: в регистр AH заносим значение 0Ah, в регистр AL – ASCII -код символа char_ctr, в регистр BH – номер страницы 0, в регистр CX – число символов 1, после чего вызывается прерывание 10h.
3. Загружаем из стека все регистры общего назначения.
4. Выполняем возврат из подпрограммы в точку вызова.
Конец.

Теоретические сведения

BIOS позволяет переключать экран в различные текстовые и графические режимы. Режимы отличаются друг от друга разрешением (для графических) и количеством строк и столбцов (для текстовых), а также количеством возможных цветов. В текстовом (алфавитно-цифровом) режиме каждый символ в области памяти дисплея (также называемой буфером) занимает два байта:
1. байт для кода символа;
2. байт атрибутов – для определения цвета и яркости символа.
Байт атрибутов определяет свойства каждого выводимого символа. Когда программа устанавливает атрибут, он остается в установленном состоянии до следующего явного изменения.
Буквы R, G, B указывают позиции битов, соответствующих красному, зеленому и синему цветам.
· Бит 7 (BL) устанавливает атрибут мерцания (может быть заблокирован).
· Биты 6 – 4 определяют цвет фона символа.
· Бит 3 (I) устанавливает для символа нормальную (0) или повышенную (1) яркость.
· Биты 2 – 0 определяют составляющие цвета символа.
Область памяти дисплея позволяет хранить данные в страницах. Страница содержит данные для одного полного экрана, а всего страниц 8, с номерами от 0 до 7. Страница 0 всегда выбрана по умолчанию и, для текстовых режимов, начинается с адреса

B800h:0000h. Страница 1 начинается с адреса B900h:0000h, страница 2 – с адреса BA00h:0000h, страница 3 – с адреса BB00h:0000h, и т.д. Вы можете работать с любой страницей в памяти, но одновременно на экран выводится только одна страница.
Все функции видеосервиса BIOS вызываются через прерывание 10h.

1.3.2.5 Функции прерывания INT 10h

Прерывание 10h предназначено для работы с дисплеем. Обработчик прерывания 10h выполняет множество функций, выбираемых по коду функции в регистре AH:

00h Установить видеорежим	0Bh Установить цветовую палитру
01h Установить размер курсора	0Ch Записать пиксель
02h Установить положение курсора	0Dh Прочитать пиксель
03h Вернуть состояние курсора	0Eh Записать в режиме телетайпа
05h Установить активную страницу	0Fh Получить текущий видеорежим
06h Прокрутить экран вверх	10h Обратиться к регистрам палитры
07h Прокрутить экран вниз	11h Обратиться к знакогенератору
08h Прочитать символ/атрибут	12h Выбрать альтернативную программу
09h Вывести символ/атрибут	13h Вывести строку символов
0Ah Вывести символ	1Bh Вернуть видеоинформацию

Рассмотрим функции прерывания 10h, которые используются в программе из упражнения 1.

Функция 00h: установка видеорежима

Используется для инициализации видеорежима исполняемой в данный момент программы. Код функции (00H) загружается в регистр AH и код режима загружается в регистр AL. Операция не возвращает значений. Установка режима также приводит к очистке экрана.
В следующем примере устанавливается стандартный цветной текстовый режим для любого типа цветного монитора:

```
MOV AH,00h ;Запрос на установку видеорежима
MOV AL,03h ;Стандартный цветной текстовый режим
INT 10h    ;Вызвать обработчик прерывания
```

Функция 02h : установка положения курсора

Эта операция в текстовом режиме устанавливает курсор в произвольное положение на экране согласно координатам в формате строка:столбец.
Для выполнения операции установите значения в следующих регистрах:
BH = номер страницы (0 по умолчанию)
DH = строка
DL = столбец
В нижеприведенном примере курсор устанавливается в строку 12, столбец 30 на странице 0:

```
MOV AH,02h ;Запросить перемещение курсора
MOV BH,00h ;Страница 0
MOV DH,12  ;Строка 12
MOV DL,30  ;Столбец 30
INT 10h    ;Вызвать обработчик прерывания
```

Функция 05h : выбор активной страницы

Позволяет выбирать страницу (в AL), выводимую на экран. Можно создать несколько различных страниц и переключаться между ними с помощью этой функции:

```
MOV AH,05h ;Выбор функции для вывода страницы
MOV AL,01h ;Страница 1
INT 10h    ;Вызвать обработчик прерывания
```

Функция 06h : прокрутка экрана вверх

Эта операция прокручивает экран вверх в выбранной области (на активной видеостранице). Выводимые строки прокручиваются вверх за пределы выводимой на экран области, а внизу прокручиваемой области появляются пустые строки. Операция не возвращает значений.

Установка ненулевого числа в AL вызывает прокрутку этого числа строк вверх. Для выполнения функции загрузите следующие значения в регистры:

AL = число строк (00 для всего экрана)

BH = атрибут

CX = начальные строка:столбец

DX = конечные строка:столбец

В текстовом режиме весь экран прокручивается вверх на одну строку:

```
MOV AH,06h   ;Запросить прокрутку вверх
MOV AL,01h   ;на одну строку
MOV CX,0000  ;от 00:00 до
MOV DX,184Fh ;24:79 (весь экран)
INT 10h      ;Вызвать обработчик прерывания
```

Функция 09h : вывод символа и атрибутов в положение курсора

Эта операция выводит заданное число символов с указанными атрибутами. Положение курсора определяет место вывода символа. Для выполнения функции задайте следующие значения в регистрах:

AL = символ ASCII

BL = атрибут

BH = номер страницы

CX = число символов

Число в регистре CX определяет, сколько раз должна повториться операция вывода символа из AL.

Число в регистре CX определяет, сколько раз должна повториться операция вывода символа из AL. В следующем примере в текстовом режиме выводятся 60 «улыбающихся лиц» (ASCII 01h) с установленными атрибутами:

```
MOV AH,09h ;Запросить вывод (в текстовом режиме)
MOV AL,01h ;Выводимый символ
MOV BH,0   ;Страница 0
MOV BL,16h ;Синий фон, коричневые символы
MOV CX,60  ;Число выводимых символов
INT 10h    ;Вызвать обработчик прерывания
```

Операция не перемещает курсор и не реагирует на символы звукового сигнала, перевода строки, возврата каретки и табуляции; вместо этого она попытается вывести их как символы ASCII. Если в текстовом режиме вывод достигает крайнего правого символа, вывод продолжится со столбца 00 в следующей строке.

Функция 0Ah : вывод символа в текущей позиции курсора

Единственное различие между этой функцией и функцией 09H состоит в том, что функция 0AH не устанавливает атрибуты, а использует их текущее значение. Операция не возвращает значений.

Функция 0Fh : получение текущего видеорежима

Считывает номер текущего видеорежима и помещает его в регистр AL, также в регистр BH помещается номер страницы выводимой на экран.

```
MOV AH,0Fh  ;Запросить видеорежим
INT 10h     ;Вызвать обработчик прерывания
```

Прямая работа с видеопамятью

Обмен данных с дисплеем с помощью BIOS и операционной системы может существенно замедлить работу некоторых приложений. Ускорить вывод графики и текста можно, записывая их напрямую в видеопамять. Запись символа и его атрибутов в видеопамять вызывает его немедленное отображение на экране.

Задание. Напишите EXE - программу, которая выводит на первую страницу видеопамяти символы звездочки так, что они образуют прямоугольник размером 15x20 и координатами левого верхнего угла 5:30.

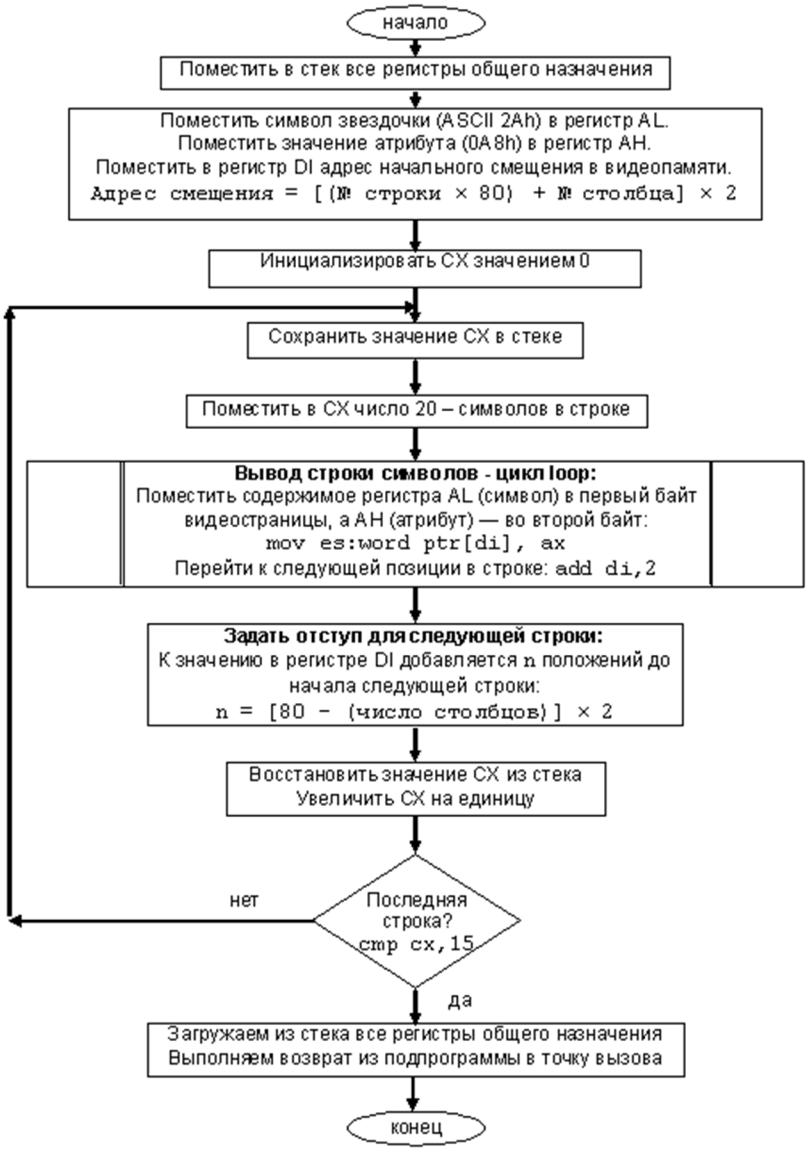
1.4 ***Порядок выполнения***

1.5 **С**труктура и алгоритм работы программы непосредственного вывода на экран

Изучите предлагаемую структуру программы и алгоритм ее работы.

```
.model small
.stack 100h
.186
.code
start:
mov ax,@data
mov ds,ax
;Код главной программы

mov ax,0b900h          ;Используя сегментный регистр ES,
mov es,ax              ;организовать запись данных в видеопамять
                       ;по адресу B900h:0000h (страница 1)
;Определить текущий видеорежим и текущую активную страницу
;(функция 0Fh прерывания int 10h) и сохранить их в стеке
;. . .
;Установить видеорежим 03 (функция 00h прерывания int 10h)
;и текущую страницу 01 (функция 05h прерывания int 10h)
;. . .
;Вызвать подпрограмму B10DISPLAY,
;которая подготавливает область вывода
call B10DISPLAY
;После завершения вывода программа ожидает нажатия клавиши
;и, восстановив исходную страницу и видеорежим, завершается
;. . .
mov ax,4C00h           ;Завершение программы
int 21h
;Определение подпрограммы
B10DISPLAY proc        ;Сохраняет символ и атрибут в области
видеопамяти
;Код подпрограммы
;. . .
ret
B10DISPLAY endp
end start              ;Конец программы
```



8. ТЕМАТИКА КУРСОВЫХ РАБОТ

Не предусмотрены

9. ПЕРЕЧЕНЬ ВОПРОСОВ К ЭКЗАМЕНУ

1. Регистры микропроцессора 8086, классификация, назначение.
2. Директива ASSUME. Директива SEGMENT.
3. Директива определения данных. Типы данных микропроцессора
4. Способы адресации
5. Команды пересылки данных
6. Арифметические команды
7. Команды сдвигов, логические команды.
8. Стек. Основные регистры работы со стеком. Основные команды работы со стеком
9. Команды LEA и XLAT

10. Команда сравнения. Команды переходов.
11. Циклы: команды, организация.
12. Строковые команды. Команды пересылки и загрузки
13. Строковые команды. Команды сравнения. Префиксы повторения.
14. Подпрограммы: описание. Команды CALL и RET.
15. Команды обработки данных в формате ASCII
16. Команды обработки данных в формате BCD
17. Математический сопроцессор: устройство, основные принципы работы, типы данных.
18. Математический сопроцессор: команды пересылки данных.
19. Математический сопроцессор: арифметические и трансцендентные команды.
20. Математический сопроцессор: команды сравнения чисел, управляющие команды.
21. Математический сопроцессор: обработка особых случаев.
22. Функции для работы с файлами прерывания 21h.
23. Работа в графическом режиме.

10. САМОСТОЯТЕЛЬНАЯ РАБОТА ОБУЧАЮЩИХСЯ

Форма обучения очная

Название разделов и тем	Вид самостоятельной работы	Объем в часах	Форма контроля
1. Общие принципы организации ЭВМ.	чтение основной и дополнительной литературы, самостоятельное изучение материала по литературным источникам;	10	Текущий контроль (проверка решения)
2. Основные элементы программирования на Ассемблере.	чтение основной и дополнительной литературы, самостоятельное изучение материала по литературным источникам;	10	Текущий контроль (проверка решения) Проверка лабораторных работ
3. Сложные типы данных в Ассемблере: массивы, строки. Организация работы с одномерными массивами и со строками.	чтение основной и дополнительной литературы, самостоятельное изучение материала по литературным источникам;	10	Текущий контроль (проверка решения) Проверка лабораторных работ
4.Сложные типы данных в Ассемблере: двумерные массивы, структуры и записи. Способы адресации	чтение основной и дополнительной литературы, самостоятельное изучение материала по литературным источникам;	10	Текущий контроль (проверка решения) Проверка лабораторных работ

операндов для работы со сложными типами данных.			
5. Процедуры и макросы в Ассемблере, назначение и использование.	чтение основной и дополнительной литературы, самостоятельное изучение материала по литературным источникам; подготовка к выполнению лабораторной работы;	14	Текущий контроль (проверка решения)

Форма обучения заочная

Название разделов и тем	Вид самостоятельной работы	Объем в часах	Форма контроля
1. Общие принципы организации ЭВМ.	чтение основной и дополнительной литературы, самостоятельное изучение материала по литературным источникам;	15	Текущий контроль (проверка решения)
2. Основные элементы программирования на Ассемблере.	чтение основной и дополнительной литературы, самостоятельное изучение материала по литературным источникам;	15	Текущий контроль (проверка решения) Проверка лабораторных работ
3. Сложные типы данных в Ассемблере: массивы, строки. Организация работы с одномерными массивами и со строками.	чтение основной и дополнительной литературы, самостоятельное изучение материала по литературным источникам;	15	Текущий контроль (проверка решения) Проверка лабораторных работ
4.Сложные типы данных в Ассемблере: двумерные массивы, структуры и записи. Способы адресации операндов для работы со сложными типами данных.	чтение основной и дополнительной литературы, самостоятельное изучение материала по литературным источникам;	15	Текущий контроль (проверка решения) Проверка лабораторных работ
5. Процедуры и макросы в Ассемблере, назначение и использование.	чтение основной и дополнительной литературы, самостоятельное изучение материала по литературным источникам; подготовка к выполнению лабораторной работы;	18	Текущий контроль (проверка решения)

11. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

а) Список рекомендуемой литературы

основная

1) Кирнос В.Н. Введение в вычислительную технику. Основы организации ЭВМ и программирование на Ассемблере [Электронный ресурс]: учебное пособие/ Кирнос В.Н.— Электрон. текстовые данные.— Томск: Томский государственный университет систем управления и радиоэлектроники, Эль Контент, 2011.— 172 с.— Режим доступа: http://www.iprbookshop.ru/13921.html .— ЭБС «IPRbooks»
2) Секаев В.Г. Основы программирования на Ассемблере [Электронный ресурс]: учебное пособие/ Секаев В.Г.— Электрон. текстовые данные.— Новосибирск: Новосибирский государственный технический университет, 2010.— 100 с.— Режим доступа: http://www.iprbookshop.ru/44986.html .— ЭБС «IPRbooks»

дополнительная

1) Аблязов Р.З. Программирование на ассемблере на платформе x86-64 [Электронный ресурс]/ Аблязов Р.З.— Электрон. текстовые данные.— Саратов: Профобразование, 2019.— 301 с.— Режим доступа: http://www.iprbookshop.ru/88005.html .— ЭБС «IPRbooks»

учебно-методическая

1) Гагарина, Л. Г. Архитектура вычислительных систем и Ассемблер с приложением методических указаний к лабораторным работам : учебное пособие / Л. Г. Гагарина, А. И. Кононова. — Москва : СОЛОН-Пресс, 2019. — 368 с. — ISBN 978-5-91359-321-4. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/94943.html (дата обращения: 09.11.2020). — Режим доступа: для авторизир. Пользователей

Согласовано:

Гл. библ. по Полякова И.Ю. _____ /_____

Должность сотрудника научной библиотеки ФИО подпись дата

б) Программное обеспечение

1. Turbo Assembler (IDE для компилятора Borland Turbo Assembler).

в) Профессиональные базы данных, информационно-справочные системы:

1. Электронно-библиотечные системы:

1.1. IPRbooks : электронно-библиотечная система : сайт / группа компаний Ай Пи Ар Медиа. - Саратов, [2020]. – URL: http://www.iprbookshop.ru. – Режим доступа: для зарегистрир. пользователей. - Текст : электронный.

1.2. ЮРАЙТ : электронно-библиотечная система : сайт / ООО Электронное издательство ЮРАЙТ. – Москва, [2020]. - URL: https://www.biblio-online.ru. – Режим доступа: для зарегистрир. пользователей. - Текст : электронный.

1.3. Консультант студента : электронно-библиотечная система : сайт / ООО Политехресурс. – Москва, [2020]. – URL: http://www.studentlibrary.ru/catalogue/switch_kit/x2019-128.html. – Режим доступа: для зарегистрир. пользователей. – Текст : электронный.

1.4. Лань : электронно-библиотечная система : сайт / ООО ЭБС Лань. – Санкт-Петербург, [2020]. – URL: http://www.studentlibrary.ru/pages/catalogue.html https://e.lanbook.com. – Режим доступа: для зарегистрир. пользователей. – Текст : электронный.

1.5. **Znanium.com :** электронно-библиотечная система : сайт / ООО Знаниум. - Москва, [2020]. - URL: http://www.studentlibrary.ru/pages/catalogue.html http://znanium.com. – Режим доступа : для зарегистрир. пользователей. - Текст : электронный.

2. КонсультантПлюс [Электронный ресурс]: справочная правовая система. /ООО «Консультант Плюс» - Электрон. дан. - Москва : КонсультантПлюс, [2020].

3. Базы данных периодических изданий:

3.1. База данных периодических изданий : электронные журналы / ООО ИВИС. - Москва, [2020]. – URL: https://dlib.eastview.com/browse/udb/12. – Режим доступа : для авториз. пользователей. – Текст : электронный.

3.2. eLIBRARY.RU: научная электронная библиотека : сайт / ООО Научная Электронная Библиотека. – Москва, [2020]. – URL: http://elibrary.ru. – Режим доступа : для авториз. пользователей. – Текст : электронный

3.3. «Grebennikon» : электронная библиотека / ИД Гребенников. – Москва, [2020]. – URL: https://id2.action-media.ru/Personal/Products. – Режим доступа : для авториз. пользователей. – Текст : электронный.

4. Национальная электронная библиотека : электронная библиотека : федеральная государственная информационная система : сайт / Министерство культуры РФ ; РГБ. – Москва, [2020]. – URL:http://www.studentlibrary.ru/pages/catalogue.html https://нэб.рф. – Режим доступа : для пользователей научной библиотеки. – Текст : электронный.

5. SMART Imagebase // EBSCOhost : [портал]. – URL: https://ebsco.smartimagebase.com/?TOKEN=EBSCO-1a2ff8c55aa76d8229047223a7d6dc9c&custid=s6895741. – Режим доступа : для авториз. пользователей. – Изображение : электронные.

6. Федеральные информационно-образовательные порталы:

6.1. Единое окно доступа к образовательным ресурсам : федеральный портал / учредитель ФГАОУ ДПО ЦРГОП и ИТ. – URL: http://window.edu.ru/. – Текст : электронный.

6.2. Российское образование : федеральный портал / учредитель ФГАОУ ДПО ЦРГОП и ИТ. – URL: http://www.edu.ru. – Текст : электронный.

7. Образовательные ресурсы УлГУ:

7.1. Электронная библиотека УлГУ : модуль АБИС Мега-ПРО / ООО «Дата Экспресс». – URL: http://lib.ulsu.ru/MegaPro/Web. – Режим доступа : для пользователей научной библиотеки. – Текст : электронный.

7.2. Образовательный портал УлГУ. – URL: http://edu.ulsu.ru. – Режим доступа : для зарегистр. пользователей. – Текст : электронный

Согласовано:

Заместитель начальника УИТиТ/ Клочкова А.В. / / 21.06.2019

Должность сотрудника УИТиТ ФИО подпись дата

12. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ:

Аудитории для проведения лекций (лекционные аудитории 3 корпуса УлГУ), семинарских занятий (лекционные аудитории 3 корпуса УлГУ), для выполнения лабораторных работ и практикумов (дисплейные классы 1 корпуса УлГУ), для проведения текущего контроля и промежуточной аттестации (лекционные аудитории 3 корпуса УлГУ).

Аудитории укомплектованы специализированной мебелью, учебной доской. Аудитории для проведения лекций оборудованы мультимедийным оборудованием для предоставления информации большой аудитории. Помещения для самостоятельной работы оснащены компьютерной техникой с возможностью подключения к сети «Интернет» и обеспечением доступа к электронной инфромационно-образовательной среде, электронно-библиотечной системе. Перечень оборудования, используемого в учебном процессе, указывается в соответствии со сведениями о материально-техническом обеспечении и оснащенности образовательного процесса, размещенными на официальном сайте УлГУ в разделе «Сведения об образовательной организации».

13. СПЕЦИАЛЬНЫЕ УСЛОВИЯ ДЛЯ ОБУЧАЮЩИХСЯ С ОГРАНИЧЕННЫМИ ВОЗМОЖНОСТЯМИ ЗДОРОВЬЯ

В случае необходимости, обучающимся из числа лиц с ограниченными возможностями здоровья (по заявлению обучающегося) могут предлагаться одни из следующих вариантов восприятия информации с учетом их индивидуальных психофизических особенностей:

– для лиц с нарушениями зрения: в форме электронного документа; индивидуальные консультации с привлечением тифлосурдопереводчика; индивидуальные задания и консультации;

– для лиц с нарушениями слуха: в печатной форме; в форме электронного документа; индивидуальные консультации с привлечением сурдопереводчика; индивидуальные задания и консультации;

– для лиц с нарушениями опорно-двигательного аппарата: в печатной форме; в форме электронного документа; индивидуальные задания и консультации.

В случае необходимости использования в учебном процессе частично/исключительно дистанционных образовательных технологий, организация работы ППС с обучающимися с ОВЗ и инвалидами предусматривается в электронной информационно-образовательной среде с учетом их индивидуальных психофизических особенностей.

Разработчик _____ доцент С.В. Липатова

подпись должность ФИО

ЛИСТ ИЗМЕНЕНИЙ

№ п/п	Содержание изменения или ссылка на прилагаемый текст изменения	ФИО заведующего кафедрой, реализующей дисциплину/выпускающей кафедрой	Подпись	Дата
1	Внесение изменений в п.п. 4.2 «Объем дисциплины по видам учебной работы (в часах)» п. 4. «Общая трудоемкость дисциплины» через слеш указать количество часов работы ППС с обучающимися для проведения занятий в дистанционном формате с применением дистанционного обучения в самой программе	Смагин А.А.		6.04.2020
2	Внесение изменений в п. 13 «Специальные условия для обучающихся с ограниченными возможностями здоровья» в самой программе	Смагин А.А.		6.04.2020
3	Внесение изменений в п.п. а) Список рекомендуемой литературы п.11 «Учебно-методическое и информационное обеспечение дисциплины» в самой программе	Смагин А.А.		6.04.2020
4	Внесение изменений в п.п. в) Профессиональные базы данных, информационно-справочные системы п. 11 «Учебно-методическое и информационное обеспечение дисциплины» в самой программе	Смагин А.А.		6.04.2020