Департамент внутренней и кадровой политики Белгородской области
Областное государственное автономное
профессиональное образовательное учреждение
«Белгородский индустриальный колледж»

Документ подписан электронной цифровой подписью.

Сертификат: 5f619c548405aee0351d9e267d24ef84c4c1cce0
Дата действия: с 25 марта 2021 г. 11:10:18 по 25 марта 2026 г. 11:10:18

Шаталов Олег Александрович
2021.09.07 08:39:32 +03'00'

Рассмотрено
цикловой комиссией
Протокол заседания №_____
от «___»_____20 г.
Председатель цикловой комиссии
_____ Третьяк И.Ю.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению лабораторных работ по

МДК.01.04 СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

по специальности

09.02.07 Информационные системы и программирование

Квалификация: программист

Разработчик:
Преподаватель
Белгородский индустриальный колледж
Ченская И.Б.

Белгород 2020 г.

Содержание

1. Пояснительная записка

1.1. Краткая характеристика МДК, цели и задачи. Место лабораторных работ в междисциплинарном курсе

МДК.01.04 Системное программирование является частью рабочей основной образовательной программы в соответствии с ФГОС по специальности СПО 13.02.11 Техническая эксплуатация и обслуживание электрического и электромеханического оборудования.

Дисциплина изучается в III-IV семестрах. В целом рабочей программой предусмотрено 60 часов на выполнение лабораторных работ, что составляет 42% от обязательной аудиторной нагрузки, которая составляет 142 часа, при этом максимальная нагрузка составляет 160 часов, из них 4 часа приходится на самостоятельную работу обучающихся.

Цель настоящих методических рекомендаций: оказание помощи обучающимся в выполнении лабораторных работ по МДК.01.04 Системное программирование, качественное выполнение которых поможет обучающимся освоить обязательный минимум содержания дисциплины и подготовиться к промежуточной аттестации в форме экзамена.

1.2. Организация и порядок проведения лабораторных работ

Лабораторные работы проводятся после изучения теоретического материала. Введение лабораторных работ в учебный процесс служит связующим звеном между теорией и практикой. Они необходимы для закрепления теоретических знаний, а также для получения практических навыков и умений. При проведении лабораторных работ задания, выполняются студентом самостоятельно, с применением знаний и умений, усвоенных на предыдущих занятиях, а также с использованием необходимых пояснений, полученных от преподавателя. Обучающиеся должны иметь методические рекомендации по выполнению лабораторных работ, конспекты лекций, измерительные и чертежные инструменты, средство для вычислений.

1.3. Общие указания по выполнению лабораторных работ

Курс лабораторных работ по МДК.01.04 Системное программирование предусматривает проведение 30 работ, посвященных изучению:

• режимов адресации;

• программирования на языке низкого уровня;

• использования ресурсов в приложениях.

При подготовке к проведению лабораторной работы необходимо:

• ознакомиться с лабораторным оборудованием;

• ознакомиться с порядком выполнения работы.

После выполнения лабораторной работы обучающийся к следующему занятию оформляет отчет, который должен содержать:

• название лабораторной работы, ее цель;

• краткие, общие сведения об изучаемом лабораторном оборудовании;

• необходимый графический материал, указанный преподавателем при выполнении лабораторной работы (принципиальная схема лабораторной установки, графики);

• данные, полученные непосредственно из проводимых опытов;

• результаты обработки данных опытов с необходимыми пояснениями;

• графический материал, отображающий полученные в ходе опытов значения измеряемых величин;

• оценку результатов испытаний.

При работе в лаборатории необходимо руководствоваться инструкциями по технике безопасности, учитывающими все специфические особенности лаборатории, такие как наличие высокого напряжения, легкодоступных для прикосновения токоведущих частей электрооборудования.

В лаборатории нельзя находиться в отсутствие преподавателя или лица, ответственного за технику безопасности.

При нахождении в лаборатории следует находиться в рабочей зоне, указанной преподавателем. С самого начала необходимо убедиться в том, что испытательный стенд находится в полностью обесточенном (отключенном) состоянии.

Перед выполнением лабораторной работы необходимо получить вводные инструкции преподавателя и внимательно ознакомиться с описанием лабораторного стенда и оборудованием.

Внимание! Включать лабораторные установки и выполнять какие-либо действия с приборами допускается ТОЛЬКО с разрешения преподавателя!

При обнаружении признаков неисправности, таких как: появление искрения, дыма, специфического запаха, следует немедленно отключить все источники электроэнергии и сообщить о случившемся преподавателю.

При возникновении реальной опасности травматизма для одного или нескольких присутствующих, участники испытания должны произвести срочное отключение лаборатории от всех источников электроэнергии выключением вводного автомата. Лаборатории должны иметь средства пожаротушения и оказания первой медицинской помощи. На первом занятии изучаются правила техники безопасности, и проводится вводный инструктаж с последующей проверкой его усвоения, о чем свидетельствует запись в журнале по технике безопасности кабинета/лаборатории, подписываемый преподавателем, проводившем инструктаж, и всеми обучающимися.

1.4. Критерии оценки результатов выполнения лабораторных работ

Критериями оценки результатов работы обучающихся являются:

– уровень усвоения обучающимся учебного материала;

– умение обучающегося использовать теоретические знания при выполнении практических задач;

– сформированность общеучебных и профессиональных компетенций:

ОК 01. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.

ОП 02.Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности.

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие.

ОК 04. Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами.

ОК 05. Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста.

ОК 06. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей.

ОК 07. Содействовать сохранению окружающей среды, ресурсосбережению, эффективно действовать в чрезвычайных ситуациях.

ОК 08. Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности.

ОК 09. Использовать информационные технологии в профессиональной деятельности.

ОК 10. Пользоваться профессиональной документацией на государственном и иностранном языках.

ПК 1.2. Разрабатывать программные модули в соответствии с техническим заданием.

ПК 1.3. Выполнять отладку программных модулей с использованием специализированных программных средств.

Критерии оценивания лабораторной работы

Оценка	**Критерии оценивания**
5	Работа выполнена в полном объеме с соблюдением необходимой последовательности проведения, содержит результаты и выводы, все записи, таблицы, рисунки, чертежи, графики выполнены аккуратно. Обучающийся владеет теоретическим материалом, формулирует собственные, самостоятельные, обоснованные, представляет полные и развернутые ответы на дополнительные вопросы.
4	Работа выполнена в полном объеме с соблюдением необходимой последовательности проведения, содержит результаты и выводы, все записи, таблицы, рисунки, чертежи, графики выполнены аккуратно. Обучающийся владеет теоретическим материалом, допуская незначительные ошибки на дополнительные вопросы.
3	Работа выполнена в полном объеме, содержит результаты и выводы, все записи, таблицы, рисунки, чертежи, графики выполнены аккуратно. Обучающийся владеет теоретическим материалом на минимально допустимом уровне, допуская ошибки на дополнительные вопросы.
2	Работа выполнена не полностью. Студент практически не владеет теоретическим материалом, допускает ошибки при ответе на дополнительные вопросы.

2. Тематическое планирование лабораторных работ

	Наименование тем	Вид и название работы студента	Количество часов на выполнение работы
Раздел 1	**Ассемблирование. Логика и организация программы**		**38**
1.4	Программирование на языке Ассемблер.	**Лабораторная работа № 1.** Знакомство с программой разработки и отладки программ на языке Ассемблера Debug.	2
		Лабораторная работа № 2. Директивы языка ассемблер.	2
		Лабораторная работа № 3. Команды пересылки данных. Стек.	2
		Лабораторная работа № 4. Форматы команд и способы адресации в реальном режиме работы процессора.	2
		Лабораторная работа № 5. Изучение среды и отладчика ассемблера.	2
		Лабораторная работа № 6. Программирование на языке Ассемблер в среде операционной системы MS DOS.	2
		Лабораторная работа № 7. Программирование на языке Ассемблер в среде операционной системы MS DOS.	2
		Лабораторная работа № 8. Поддержка ввода с клавиатуры и вывода на экран средствами BIOS и DOS.	2
		Лабораторная работа № 9. Программирование целочисленных вычислений.	2
		Лабораторная работа № 10.Программирование логических вычислений.	2
		Лабораторная работа № 11.Программирование ветвлений и итерационных циклов.	2
		Лабораторная работа № 12.Программирование массивов и циклов.	2
		Лабораторная работа № 13.Строковые функции.	2
		Лабораторная работа № 14.Процедуры.	2
		Лабораторная работа № 15.Сервисы DOS для работы с файловой системой.	2
		Лабораторная работа № 16.Разработка подпрограмм и программных прерываний средствами языка Ассемблер.	2
		Лабораторная работа № 17.Разработка подпрограмм и программных прерываний средствами языка Ассемблер.	2
		Лабораторная работа № 18.Копирование файлов с использованием Win32.	2
		Лабораторная работа № 19.Вывод списка файлов и их атрибутов в заданном каталоге.	2
Раздел 2	**Работа с Windows**		**22**

2.1	Консольные и оконные приложения Windows	**Лабораторная работа № 20.** Копирование нескольких файлов в стандартный вывод.	2
		Лабораторная работа № 21. Последовательная обработка файлов с использованием отображения.	2
		Лабораторная работа № 22. Использование динамических библиотек для создания приложений.	2
		Лабораторная работа № 23. Многопроцесная обработка данных.	2
		Лабораторная работа № 24. Расширенный ввод-вывод с процедурами завершения.	2
		Лабораторная работа № 25. Рисование графических фигур на экране монитора.	2
		Лабораторная работа № 26. Создание приложения Windows.	2
		Лабораторная работа № 27. Примеры использования классов при создании приложений Windows.	2
		Лабораторная работа № 28. Приложения, обрабатывающие клавиатурные сообщения, сообщения от драйвера "мыши" и таймера.	2
		Лабораторная работа № 29. Использование ресурсов в приложениях Windows.	2
		Лабораторная работа № 30. Использование органов управления ОС Windows.	2
		Итого:	**60**

3. Содержание лабораторных работ

Лабораторная работа №1

Тема: Знакомство с программой разработки и отладки программ на языке Ассемблера Debug»

Цель работы: *ознакомиться с программой разработки и отладки программ на языке Ассемблера, а также изучить основные команды отладчика.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

DEBUG – это системная программа, позволяющая выполнять просмотр и изменение состояний процессора и памяти компьютера, побайтное тестирование и побайтную обработку дисковых файлов, что обеспечивает возможность выполнения отлаживаемых программ небольшими порциями. При этом программа выполняется под "наблюдением" отладчика.

Таким образом, основное назначение этой программы – отладка программ на уровне машинных кодов и языка ассемблера. Однако возможности, предоставляемые этой программой, делают ее удобным инструментом для изучения организации персональных компьютеров.

Запуск отладчика

Чтобы запустить отладчик, в командной строке Windows (Меню Пуск – Выполнить) вводится команда debug (см. рис. 1).

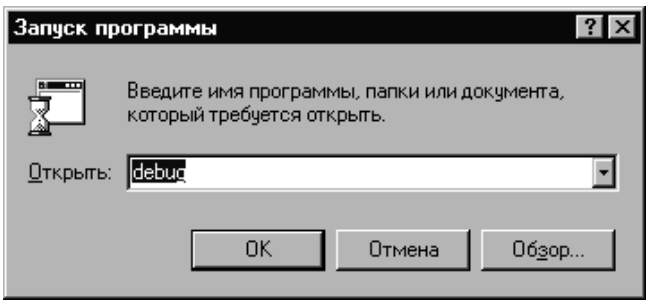


Рис. 1. Диалог "Запуск программы debug"

После нажатия кнопки "*OK*" диалога запуска отладчик загружается в память машины и переходит в режим ожидания ввода команды.

Команды программы DEBUG

DEBUG – это программа, работающая по принципу "команда – действие", т.е., чтобы произвести некоторую операцию отладчик должен получить соответствующую команду. В качестве сигнала о готовности принять команду, отладчик посылает на экран стандартный запрос – дефис (-).

Общими замечаниями по вводу команд отладчика является следующее:

– все команды начинаются с буквы, заглавной или строчной;

– большая часть команд требует введения дополнительных параметров, часть из которых является необязательными;

– если два подряд расположенных параметра являются числами, то они разделяются пробелом или запятой (в противном случае параметры можно не отделять один от другого);

– все числа должны вводиться в шестнадцатеричном представлении;

– некоторые команды принимают в качестве параметра адрес, который может вводиться в двух формах: полный логический адрес – два шестнадцатеричных числа, записанные через двоеточие (первое число – сегментная компонента логического адреса, второе число – смещение) и короткий адрес – одно шестнадцатеричное число (смещение);

– при указании полного логического адреса допускается в качестве сегментной компоненты приводить имя сегментного регистра, из которого данная компонента будет выбираться.

С точки зрения изучения организации ЭВМ, набор команд, предлагаемый отладчиком, является избыточным. Минимально необходимый набор включает команды:

– (A)SSEMBLE – ассемблирование;

– (U)NASSEMBLE – дизассемблирование;

– (E)NTER – ввод данных в память;

– (D)UMP – вывод содержимого участка памяти на экран;

– (R)EGISTER – просмотр и изменение содержимого регистров;

– (T)RACE – пошаговое выполнение программы;

– (N)AME – задание имени файла программы;

– (L)OAD – загрузки файла в память;

– (W)RITE – запись области памяти в файл;

– (Q)UIT – выход из отладчика.

Перечисленные команды представляют для нас наибольший интерес. Рассмотрим их подробнее.

Команда просмотра и изменения содержимого регистров

Команда REGISTER (*r* или *R*) выводит на экран и корректирует значения регистров и флагов состояния процессора. Эта команда также выдает информацию о следующей выполняемой команде:

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0958 ES=0958 SS=0958 CS=0958 IP=0100    NV UP DI PL NZ NA PO NC
0958:0100  0000        ADD             [BX+SI],AL            DS:0000=CD
-
```

С помощью "*r*" можно изменить значение регистра. В этом случае в командной строке указывается его имя. Значение регистра выводится на экран. Теперь можно вводить новое число. Чтобы сохранить старое значение регистра, нажмите *Enter*.

```
-r CX
```

```
CX 0000
:245D
-r
AX=0000 BX=0000 CX=245D DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0958 ES=0958 SS=0958 CS=0958 IP=0100    NV UP DI PL NZ NA PO NC
0958:0100  0000        ADD            [BX+SI],AL            DS:0000=CD
-
```

Команда "*rf*" выводит на экран флаги состояния процессора. Получив значения флагов, их можно изменить. Для этого вводится одно или несколько новых значений.

Мнемонические обозначения состояний флагов приведены в табл. 1.

Таблица 1

Мнемонические обозначения состояний флагов

Флаг	Установлен	Сброшен
Переполнение (есть/нет)	OV	NV
Направления (увеличение/уменьшение)	DN	UP
Прерывания (разрешение/запрещение)	EI	DI
Знака (минус/плюс)	NG	PL
Нуля (да/нет)	ZR	NZ
Дополнительного переноса (да/нет)	AC	NA
Паритета (чет/нечет)	PE	PO
Переноса (да/нет)	CY	NC

Символьные значения вводятся в любом порядке через пробел или вообще без разделителя. Установим, например, значения флагов переполнения, знака и переноса:

```
-rf
 NV UP DI PL NZ NA PO NC     -OV NG CY <- Подчеркнутое вводит пользователь
-r
AX=0000 BX=0000 CX=245D DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0958 ES=0958 SS=0958 CS=0958 IP=0100    OV UP DI NG NZ NA PO CY
0958:0100  CD20        INT       20
```

Команда пошагового выполнения программы

Команда TRACE (*t* или *T*) – трассировка осуществляет пошаговое выполнение программы в машинном коде. При трассировке после выполнения каждой команды производится останов работы программы и на экран выводятся регистры и флаги состояния процессора. Полученная картинка аналогична картинке, получаемой с помощью команды REGISTER. Разница заключается только в том, что при введении TRACE перед появлением картинки, выполняется одна команда отлаживаемой программы.

Проиллюстрируем работу TRACE на примере нашей программы. Если она не загружена в память, то запустим DEBUG и введем:

```
-e CS:0100 B0 2A BF 00 02 B9 1D 00 FC F2 AA B0 24
```

Чтобы узнать адрес программы, введем команду REGISTER:

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
```

```
DS=0976 ES=0976 SS=0976 CS=0976 IP=0100    NV UP DI PL NZ NA PO NC
0976:0100  B001       MOV          AL,2A
```

При введении "*t*" выполняется команда по адресу CS:IP. После этого на экран выводятся регистры и флаги состояния:

```
-t
AX=002A BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0976 ES=0976 SS=0976 CS=0976 IP=0102    NV UP DI PL NZ NA PO NC
0976:0102  BF0002      MOV          DI,0200
-
```

В командной строке TRACE можно указать адрес выполняемой команды. В этом случае после *t* набирается знак равенства (=) и нужный адрес. Если указан короткий адрес, то адрес сегмента выбирается из регистра *CS*:

```
-t=0100
AX=002A BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0200
DS=0976 ES=0976 SS=0976 CS=0976 IP=0102    NV UP DI PL NZ NA PO NC
0976:0102  BF0002      MOV          DI,0200
-
```

В этом случае выполнена команда по адресу CS:0100. Адрес следующей команды находится в регистрах *CS*:*IP*. Он равен 0976:0102.

Одной командой TRACE можно одновременно трассировать несколько команд отлаживаемой программы. Для этого при введении "*t*" просто указывается их количество. После выполнения каждой команды на экране появляется картинка с содержимым регистров и флагов состояния. При заполнении экрана новые данные выводятся в нижней его части, сдвигая данные в верхней части за пределы экрана. Чтобы остановить движение данных вдоль экрана, нажимаются клавиши *Ctrl-NumLock*. Чтобы возобновить движение, нажимается любая клавиша.

При нажатии *Ctrl-C* трассирование прекращается и на экране появляется стандартный запрос отладчика.

Пример:

```
-t6
AX=002A BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0200
DS=0976 ES=0976 SS=0976 CS=0958 IP=0105    NV UP DI PL NZ NA PO NC
0976:0105  B91D00      MOV       CX,001D
```

Команда задания имени файла программы

Команда NAME (*n* или *N*) присваивает имя обрабатываемому файлу. Затем этот файл загружается в память командой LOAD или записывается на диск командой WRITE. (LOAD и WRITE рассматриваются ниже.)

Чтобы идентифицировать файл, наберите "*n*" и, через пробел – имя файла. Воспользуемся NAME, чтобы присвоить нашей программе имя "mytest.pro":

```
-n mytest.pro
```

Команда загрузки файла в память

Загрузка файла в память осуществляется, если в командной строке DEBUG указать имя файла. Другой способ – использование *команды LOAD* (*l* или *L*).

При использовании команды LOAD необходимо специфицировать файл с помощью команды NAME.

В командной строке LOAD можно указать начальный адрес, по которому загружается файл. Если указан короткий адрес, то адрес сегмента выбирается из регистра *CS*. При отсутствии начального адреса, загрузка производится по адресу *CS*:0100.

После загрузки отладчик запоминает количество занятой файлом памяти (в байтах) в регистрах *BX* (старшее слово) и *CX* (младшее слово).

К примеру, загрузим в память файл "mytest.pro" по адресу *CS*:0100:

```
-n mytest.pro
-L
-r
AX=0000 BX=0000 CX=00CF DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0958 ES=0958 SS=0958 CS=0958 IP=0100    NV UP DI PL NZ NA PO NC
0958:0100  2A2A        SUB             CH,[BP+SI]            SS:0000=CD
-
```

В регистрах *BX* и *CX* находится значение 207 (000000CF). Это значит, что файл занял 207 байт. Тот же результат можно получить при введении спецификации файла в командной строке команды старта отладчика ("debug mytest.pro").

Команда записи области памяти в файл

Команда WRITE (*w* или *W*) переписывает на диск данные, выбирая их из памяти. При этом спецификация создаваемого файла должна задаваться с помощью команды NAME.

Перед введением команды WRITE в регистры *BX* и *CX* записывается размер занимаемой файлом памяти в байтах (шестнадцатеричное число, занимающее 4 байта). Поэтому перед записью необходимо проверить содержимое этих регистров (с помощью REGISTER).

В командной строке WRITE можно указать начальный адрес памяти, по которому производится чтение данных с последующей записью их на диск. Если указан короткий адрес, то адрес сегмента выбирается из регистра *CS*.

Если начальный адрес не указан, то запись производится, начиная с адреса *CS*:0100.

Команда выхода из отладчика

Чтобы выйти из отладчика и передать управление операционной системе, на его стандартный запрос вводится команда *q*:

```
-q
```

Задание. Изучить основные команды, выполнив все примеры (синего цвета).

Задание 2. В соответствии со своим вариантом выполнить задание в DEBUG и проверить результат в ячейках.

1. *MOV BP,200*
2. *MOV BP,30*
3. *MOV DI,AX*
4. *MOV BP,AX*
5. *MOV SI,–20*
6. *MOV BX,AX*
7. *MOV DI,18*
8. *MOV SI,AX*
9. *MOV SI,40*
10. *MOV AX,120*

Контрольные вопросы:

1. Назначение программы DEBUG?
2. Что означает дизассемблирование?
3. Какие режимы работы процессора вы знаете?
4. Как записывается команда ассемблирования?
5. Какая команда служит для отображения на экране содержимого участка памяти?
6. Какая команда переписывает на диск данные, выбирая их из памяти?

Лабораторная работа №2

Тема: Директивы языка Ассемблера

Цель работы:

- ознакомление со структурой программы на ассемблере;
- дать понятие о директивах определения данных;
- директивы DATA, CODE, Mode.

Ход работы:

1. **Изучить теоретическую часть**
2. **Выполнить задание в соответствии с указаниями**
3. **Ответить на контрольные вопросы**
4. **Предъявить преподавателю результаты работы: проект и исходный код**
5. **Оформить отчет в соответствии с ходом работы**

Теоретическая часть:

Программа на ассемблере представляет собой совокупность блоков памяти, называемых сегментами памяти. Программа может состоять из одного или нескольких таких блоков-сегментов. Каждый сегмент содержит совокупность предложений языка, каждое из которых занимает отдельную строку кода программы.

Предложения ассемблера бывают четырех типов:

- команды или инструкции, представляющие собой символические аналоги машинных команд. В процессе трансляции инструкции ассемблера преобразуются в соответствующие команды системы команд микропроцессора;
- макрокоманды — оформляемые определенным образом предложения текста программы, замещаемые во время трансляции другими предложениями;
- директивы, являющиеся указанием транслятору ассемблера на выполнение некоторых действий. У директив нет аналогов в машинном представлении;
- строки комментариев, содержащие любые символы, в том числе и буквы русского алфавита. Комментарии игнорируются транслятором.

Для простых программ, содержащих по одному сегменту для кода, данных и стека, хотелось бы упростить ее описание. Для этого в трансляторы ввели возможность использования упрощенных директив сегментации. Но здесь возникла проблема, связанная с тем, что необходимо было как-то компенсировать невозможность напрямую управлять размещением и комбинированием сегментов. Для этого совместно с упрощенными директивами сегментации стали использовать директиву указания модели памяти MODEL, которая частично стала управлять размещением сегментов и выполнять функции директивы ASSUME (поэтому при использовании упрощенных директив сегментации директиву ASSUME можно не использовать). Эта директива связывает

сегменты, которые в случае использования упрощенных директив сегментации имеют предопределенные имена, с сегментными регистрами (хотя явно инициализировать ds все равно придется).

Синтаксис директивы MODEL показан на рис. 1.

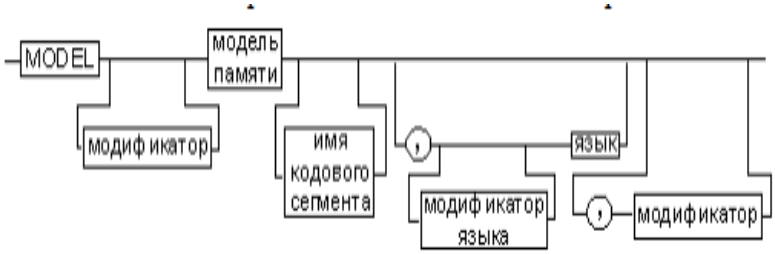


Рис. 1 Синтаксис директивы MODEL

.COM (англ. *command*) — расширение файла, использовалось в некоторых операционных системах в различных целях.

В системах DOS и в 8-битной CP/M COM-файл — простой тип исполняемого файла, размер которого не может превышать 65280 байт (на 256 байт меньше размера 16-битного сегмента — 2^{16}–256 байт). COM-файлы для DOS можно выполнять также на эмуляторах, например, в среде Windows.

Кроме COM-файлов DOS поддерживает файлы в формате EXE. Тип файла определяется при запуске автоматически (в формате EXE в начале файла имеется специальная сигнатура), независимо от расширения.

Для запуска .COM-программы MS-DOS выделяет сегмент памяти, устанавливает на него все сегментные регистры, в первых 256 байтах строит PSP, содержимое COM-файла без изменений загружается следом за ним и запускается с первого байта (то есть с адреса 256), предварительно установив указатель стека в конец сегмента.

Модель памяти, используемую COM-программами, когда код программы, все ее данные, PSP и стек расположены в одном сегменте, компиляторы высокоуровневых языков называют TINY (англ. *tiny* — крохотная).

COM-программы обычно являются небольшими приложениями, системными утилитами или небольшими резидентными программами.

Примечание: **Вам нужно выполнить все задания! Для этого необходимо сделать шаги:**

1. В текстовом редакторе создаем файл, например primer1.asm и записываем в него код (без номеров строк).
2. Запускаем командную строку, с помощью команды «cd..» переходим в каталог C:\TASM.
3. Компилируем программу: `C:\TASM>tasm primer1.asm`
4. Компонуем программу: `C:\TASM>tlink primer1.obj`
5. Запуск и тестирование программы: `C:\TASM>primer1.exe`

Задание. Создайте файл с именем Lab2.asm.

```
.Model Small      ;Модель памяти ближнего типа
.Stack 100h       ;Определяет стек размером 100h
.Data             ;Начало сегмента данных
Hello DB "Laboratornaja №2 !$'  ;Зарезервировали память для переменной HELLO
.Code             ;Начало сегмента кода
Start:
mov ax,@DATA  ;Формирование адреса сегмента
mov ds,ax         ;данных
lea DX,Hello      ;Установить в DX адрес переменной HELLO
mov ah,09h        ;Функция DOS вывода строки
int 21h           ;Вывод строки на экран
mov ax,4C00h      ;Функция DOS завершения
int 21h           ;программы
END start         ;конец программы
```

Задание 2. Введите текст следующей программы:

```
; Пример оформления программы в виде com-файла
.MODEL SMALL
.CODE
org 100h
begin:
jmp start
Hello DB 'Hello!$'
start: LEA DX,Hello
mov ah,09h
int 21h
mov ah,4ch
mov al,00h
int 21h
END begin
```

Контрольные вопросы:

7. Какую структуру имеет программа на языке ассемблера?
8. Какого вида предложения бывают в исходном коде на ассемблере?
9. Опишите назначения директив Assume и Model?
10. Модели памяти директива Model.
11. Расскажите разницу между директивами .Code, .Data и .Stack?
12. Какие директивы определения данных существуют, их отличия и применение?
13. Создание .com и .exe, их структурное отличие на примере сегментов памяти?

Лабораторная работа №3

Тема: Команды пересылки данных. Стек

Цель работы:

- изучение режимов адресации;
- изучение правил адресации при использовании регистров;
- дать понятие о директивах определения данных;
- понятие о стеке.

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы

4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Во всех 16-битовых ЭВМ принимаются какие-либо меры для расширения адресного пространства памяти, т.к 16-битовый адрес позволяет адресоваться только к 64кб. Память разбивается на сегменты размером до 64кб. По умолчанию каждый сегмент начинается с границы параграфа (фрагмента памяти размером в 16 б). Физический адрес в памяти складывается из начального адреса сегмента и 16-битового смещения (исполнительного адреса, ЕА) в пределах сегмента. Для получения физического начального адреса (ФА) содержимое сегментного регистра (начальный номер параграфа) умножается на 16, т.е дополняется справа четырьмя нулями:

ФА=ННП*16+смещение.

КОМАНДЫ ПЕРЕСЫЛКИ ДАННЫХ

Описание действия команды в таблице, как правило, очень кратко и схематично и неформально раскрывается в тексте, идущей за таблицей.

Переместить	`Mov dst,src`	DST ← SRC
(**MOV**e data — перемещение данных)		флаги не изменяются

СТЕК

В ПК имеются специальные команды работы со стеком, т.е областью памяти, доступ к элементам которой осуществляется по принципу «последним записан - первым считан». Но для того, чтобы можно было воспользоваться этими командами, необходимо соблюдение ряда условий.

Адресация с примерами

В архитектуре МП 8086/8088 адрес любого байта задается двумя 16-битовыми словами - сегментом и смещением. При формировании 20-разрядного полного адреса, необходимого для адресации в пределах 1 Мбайт, сегмент сдвигается влево на 4 разряда (умножается на 16) и складывается со смещением. Поскольку емкость 16-разрядного смещения составляет 65536 значений, в пределах одного сегмента можно адресовать до 64 Кбайт.

Архитектура МП позволяет использовать семь различных способов адресации.

Регистровая

Извлекает операнд из регистра или помещает его в регистр. Примеры:

mov ax,bx {Извлекаем из BX и помещаем в AX}

add cx,ax {Содержимое AX прибавляем к CX}

push ex {Заталкиваем в стек содержимое CX}

Непосредственная

Операнд (8- или 16-разрядная константа) содержится непосредственно в теле команды. Примеры:

mov ax,100 {Загружаем в AX значение 100}

add ax,5 {К содержимому AX прибавляем 5}

mov cx,$FFFF {Помещаем в CX значение 65535}

Прямая

Смещение операнда задается в теле программы и складывается с регистром DS; например:

var

X: Word; B: Byte;

.....

mov ax,X {Пересылаем значение переменной X регистр AX}

add ah,B {К содержимому регистра AH прибавляем значение переменной B}

mov X,ax {Пересылаем содержимое регистра AX, в область памяти переменной X}

Косвенная регистровая

Исполнительный адрес операнда (точнее, его смещение) содержится в одном из регистров BX, BP, SI или DI. Для указания косвенной адресации этот регистр должен заключаться в квадратные скобки, например:

mov ax,[bx] {Содержимое 16-разрядного слова, хранящегося в памяти по адресу DS:BX,пересылаем в регистр AX};

Каждый из регистров BX...DI по умолчанию работает со своим сегментным регистром:

DS:BX, SS:BP, DS:SI, ES:DI

Допускается явное указание сегментного регистра, если он отличается от умалчиваемого, например:

mov ax,es:[bx]

Адресация по базе

Базовый регистр BX (или BP) содержит базу (адрес начала некоторого фрагмента памяти), относительно которой ассемблер вычисляет смещение, например:

mov ax,[Ьx]+10 {Загружаем в AX 10-й по счету байт от начала базы памяти по адресу DS-.BX};

Индексная адресация

Один из индексных регистров SI или DI указывает положение элемента относительно начала некоторой области памяти. Пусть, например, AOB - имя массива значений типа Byte. Тогда можно использовать такие фрагменты:

mov si,15 {Помещаем в SI константу 15}

mov ah,AOB[si] {Пересылаем в AH 16-й по порядку байт от начала массива}

mov si,0

mov AOB[si],ah {Пересылаем полученное в самый первый элемент массива}

Адресация по базе с индексированием

Вариант индексной адресации для случая, когда индексируемая область памяти задается своей базой. Например:

mov ax,[bx][si]

Этот тип адресации удобен при обработке двумерных массивов. Если, например, AOB есть массив из 10x10 байт вида

var

AOB: array [0..9,0..9] of Byte;

то для доступа к элементу AOB [2,3] можно использовать такой фрагмент

mov bx,20 {База строки 2}

mov si,2 {Номер 3-го элемента}

mov ax,AOB[bx] [si] {Доступ к элементу}

Задание. Подготовить измененное содержимое ds, es, ss. Написать нижеприведенную программу с именем lab3.asm, сделать исполняемый файл, и проследить за работой в турбоотладчике.

На основе работы программы в таблице 2, в графы 2 и 3 занести ожидаемые значения операндов:

Таблица 2

Оператор	Операнд-приемник	
	до выполнения	После выполнения
1	2	3

```
SSEG        segment    para stack   'stack'
            Db     1,2,3,4,5,6,7,8,9,128 dup(0Ah)
SSEG        ends
DSEG        segment    para public   'data'
B_TAB       db     1Ah,2Bh,3Ch,4Dh,5Eh,6Fh,7Ah,8Bh
W_TAB       dw     1A2Bh,3C4Dh,5E6Fh,7A8Bh
B_TAB1      db     0Ah,8 dup(1)
W_TAB1      dw     8 dup(1)
DSEG        ends
ESEG        segment
W_TAB2      dw         11h,12h,13h,14h,15h,16h,17h,18h
ESEG        ends
CSEG        segment para public 'code'
PROG        proc   far
            assume         ds:DSEG,cs:CSEG,ss:SSEG,es:ESEG
            push   ds
            mov    ax,0
            push   ax
;инициализация сегментных регистров
            mov    ax,dseg
            mov    ds,ax
            mov    ax,eseg
            mov    es,ax
;непосредственная (операнд-источник)
            mov    al,-3            ;расширение знака
            mov    ax,3
            mov    B_TAB,-3
            mov    W_TAB,-3
            mov    ax,2A1Bh
;регистровая
            mov    bl,al
```

```
                mov   bh,al
                sub   ax,bx
                sub   ax,ax
;прямая
                mov   ax,W_TAB
                mov   ax,W_TAB+3
                mov   ax,W_TAB+5
                mov   al,byte ptr W_TAB+6
                mov   al,B_TAB
                mov   al,B_TAB+2
                mov   ax,word ptr B_TAB
                mov   es:W_TAB2+4,ax
;косвенная
                mov   bx,offset B_TAB
                mov   si,offset B_TAB+1
                mov   di,offset B_TAB+2
                mov   dl,[bx]
                mov   dl,[si]
                mov   dl,[di]
                mov   ax,[di]
                mov   bp,bx
                mov   al,[bp]                   ;какой сегмент?
                mov   al,ds:[bp]
                mov   al,es:[bx]
                mov   ax,cs:[bx]
; базовая
              mov   ax,[bx]+2           ;основная форма
              mov   ax,[bx]+4           ;проверьте допустимость других
              mov   ax,[bx+2]
              mov   ax,[4+bx]
              mov   ax,2+[bx]
              mov   ax,4+[bx]
              mov   al,[bx]+2
              mov   bp,bx                       ;другой базовый регистр
              mov   ax,[bp+2]                   ;откуда содержимое ax?
              mov   ax,ds:[bp]+2                ;попробуем переназначить
                                                ;сегментный регистр
              mov   ax,ss:[bx+2]
;индексная
              mov   si,2                        ;загрузка индекса
              mov   ah,B_TAB[si]                      ;основная форма
              mov   al,[B_TAB+si]               ;проверьте другие
              mov   bh,[si+B_TAB]
              mov   bl,[si]+B_TAB
              mov   bx,es:W_TAB2[si]
              mov   di,4
              mov   bl,byte ptr es:W_TAB2[di]
              mov   bl,B_TAB[si]
;базовая индексная
              mov   bx,offset B_TAB             ;загрузка базы
```

```
                mov   al,3[bx][si]              ;основная форма
                mov   ah,[bx+3][si]
                mov   al,[bx][si+2]
                mov   ah,[bx+si+2]
                mov   bp,bx
                mov   ah,3[bp][si]              ;из какого сегмента?
                mov   ax,ds:3[bp][si]
                mov   ax,word ptr ds:2[bp][si]
                ret
PROG    endp
CSEG    ends
end     PROG
```

Вы можете выполнить задание с помощью программы Emu8086. Дополнительная информация в файле !Emu8086.docx на сетевом диске.

Контрольные вопросы:

1. Сколько режимов адресации существует, в чем разница?
2. Приведите общий формат команды определения данных программ на Ассемблере.
3. Назначение команды POP и PUSH, приведите общий формат команды, приведите пример использования этой команды.
4. Назначение оператора PTR?
5. Для чего нужен стек в программе?
6. Способы определения сегментов памяти?

Лабораторная работа №4

Тема: Формат команд и способы адресации в реальном режиме работы процессора

Цель работы: *изучить способы адресации 16-разрядного режима работы процессора и их представление в кодах команд.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Данные, над которыми процессор выполняет свои операции, могут располагаться как в регистрах, так и в оперативной памяти компьютера. Для работы с памятью используются шина адреса и шина данных (см. рис. 1).

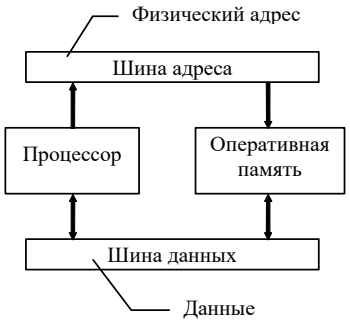


Рис. 1. Организация взаимодействия процессора и оперативной памяти

Физически память устроена таким образом, что возможно обращение к отдельным байтам, 16-разрядным словам (два смежных байта памяти), 32-разрядным двойным словам (четыре смежных байта памяти). Максимальный размер данных определяется разрядностью шины данных и режимом работы процессора. В современных персональных компьютерах шина адреса 32-разрядная, но в реальном (16-разрядном) режиме работы процессора максимальный размер обрабатываемых данных – 16 бит, т.е. слово.

Разрядность шины адреса определяет максимальный объем оперативной памяти. В современных персональных компьютерах шина адреса так же 32-разрядная, и, следовательно, процессор имеет возможность одновременно обращаться к 2^{32} = 4 Гбайт оперативной памяти. Однако в реальном режиме работы из шины адреса используется только 20 младших разрядов. Таким образом объем адресуемой памяти в реальном режиме работы процессора составляет 2^{20} = 1 Мбайт.

Так как адреса принято записывать в шестнадцатеричной системе счисления, то мы можем записать диапазон физических адресов для 20-разрядной шины адреса следующим образом:

00000h ≤ [физический адрес] ≤ *FFFFFh*.

Форматы операндов

Режим адресации	Формат операнда	Сегментный регистр
Регистровый	AL, AH, AX BL, BH, BX CL, CH, CX DL, DH, DX	Не используется

	SP BP SI DI	
Непосредственный	Данное	Не используется
Прямой	Сдвиг Метка	DS DS
Косвенный	[BX] [BP] [DI] [SI]	DS SS DS DS
По базе	[BX + сдвиг] [BP + сдвиг]	DS SS
Индексная	[DI + сдвиг] [SI] + сдвиг	DS DS
Базово-индексная	[BX + SI + сдвиг] [BX + DI + сдвиг] [BP + SI + сдвиг] [BP + DI + сдвиг]	DS DS SS SS

Задание. В соответствии со своим вариантом опробовать в DEBUG и пояснить команды. Осуществить ассемблирование и дизассемблирование указанных преподавателем команд.

Варианты заданий

1. *MOV BX,100*
   MOV BP,[BX]
   MOV DX,[BP + 2]
   MOV AX,DX
   ADD AX,[BX + 4]

2. *MOV AX,200*
   ADD AX,[BX]
   MOV DI,AX
   SUB AX,[BX + DI + 2]
   CMP AX,[DI]

3. *MOV DI,4*
   MOV BX,100
   MOV AX,[BX + DI + 4]
   MOV CL,AL
   SUB CX,[BX + SI]

4. *MOV BP,70*
   ADC BP,[BP]
   MOV AX,BX
   ADD AX,[BP + 7]
   MOV DX,[BP + 4]

5. *MOV AX,200*
   MOV BX,AX
   MOV BX,[BX + 4]
   MOV CX,[BX + 6]
   MOV DX,[BX + 8]

6. *ADD BX,[BX + 2]*
   MOV DI,8
   MOV DX,[BX + DI + 8]
   SUB BX,DX
   MOV AX,[DI]

Контрольные вопросы:

1. Организация взаимодействия процессора и оперативной памяти?
2. Как записывается логический адрес?
3. Какие режимы адресации существуют?

4. Формат команд?
5. Формирование физического адреса?

Лабораторная работа №5

Тема: Изучение среды и отладчика ассемблера

Цель работы: *ознакомление с общими принципами построения программы на языке ассемблер.*

Ход работы:

1. Изучить теоретическую часть (дополнительная информация в файле Структура COM и EXE файлов.docx)
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

1.1 Структура ассемблерной программы

Каждый язык программирования имеет свои особенности. Язык ассемблера - не исключение. Традиционно первая программа выводит приветственное сообщение на экран ‘Hello World’.

В отличие от многих современных языков программирования в ассемблерной программе каждая команда располагается на ОТДЕЛЬНОЙ СТРОКЕ. Нельзя разместить несколько команд на одной строке. Не принято, также, разбивать одну команду на несколько строк.

Язык ассемблера является РЕГИСТРОНЕЧУВСТВИТЕЛЬНЫМ.Т. е. в большинстве случаев нет разницы между большими и малыми буквами. Команда может быть ДИРЕКТИВОЙ - указанием транслятору. Они выполняются в процессе превращения программы в машинный код. Многие директивы начинаются с точки. Для удобства чтения программы они обычно пишутся БОЛЬШИМИ БУКВАМИ. Кроме директив еще бывают ИНСТРУКЦИИ - команды процессору. Именно они и будут составлять машинный код программы.

1.2 Процесс обработки программы на языке ассемблера

Весь процесс технического создания ассемблерной программы можно разбить на 4 шага (исключены этапы создания алгоритма, выбора структур данных и т.д.).

Набор программы в текстовом редакторе и сохранение ее в отдельном файле. Каждый файл имеет имя и тип, называемый иногда расширением. Тип в основном используется для определения назначения файла. Например, программа на C имеет тип C, на Pascal - PAS, на языке ассемблера - ASM.

Обработка текста программы транслятором. На этом этапе текст превращается в машинный код, называемый объектным. Кроме того, есть возможность получить листинг программы, содержащий кроме текста программы различную дополнительную информацию и таблицы, созданные транслятором. Тип объектного файла - OBJ, файла листинга - LST. Этот этап называется ТРАНСЛЯЦИЕЙ.

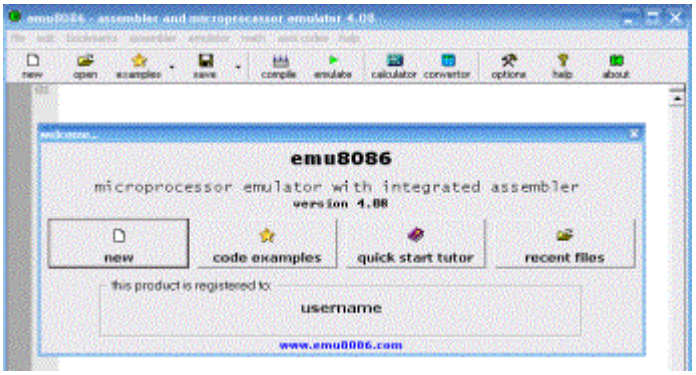
Обработка полученного объектного кода компоновщиком. Тут программа "привязывается" к конкретным условиям выполнения на ЭВМ. Полученный машинный код называется выполняемым. Кроме того, обычно получается карта загрузки программы в ОЗУ. Выполняемый файл имеет тип EXE, карта загрузки - MAP. Этот этап называется КОМПОНОВКОЙ или ЛИНКОВКОЙ.

Запуск программы. Если программа работает не совсем корректно, перед этим может присутствовать этап ОТЛАДКИ программы при помощи специальной программы - отладчика. При нахождении ошибки приходится проводить коррекцию программы, возвращаясь к шагу 1. Таким образом, процесс создания ассемблерной программы можно изобразить в виде следующей схемы. Конечной целью, напомним, является работоспособный выполняемый файл HELLO. EXE.

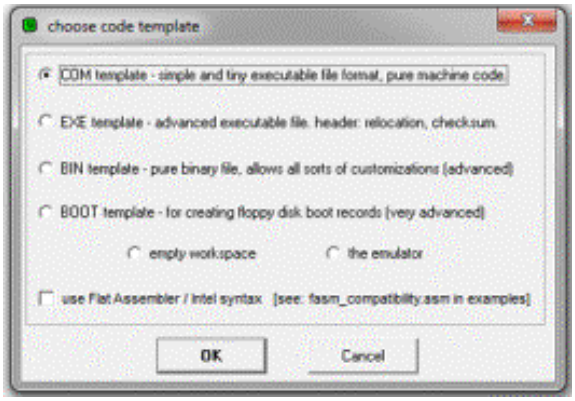
Задание. Работа с эмулятором Emu8086

Emu8086 сочетает в себе мощный редактор исходного кода, ассемблер, дизассемблер, программный эмулятор (виртуальный ПК) с отладчиком и поэтапное обучение. Эта программа компилирует исходный код и выполняет его с помощью эмулятора шаг за шагом.

1. Запустите эмулятор и щелкните на кнопку new



2. Выберите тип исполняемого файла:



Директивы, определяющие тип исполнимого файла:

#MAKE_COM#

#MAKE_BIN#

#MAKE_BOOT#

#MAKE_EXE#

Вы можете вставить эти директивы в исходный код для определения нужного вам типа исполнимого файла.

Описание типов исполнимых файлов:

#MAKE_COM# - самый старый и самый простой формат исполнимого файла. Такие файлы загружаются с префиксом 100h (256 байтов).

#MAKE_EXE# - более "продвинутый" формат исполнимого файла. Не ограничены размер и количество сегментов.

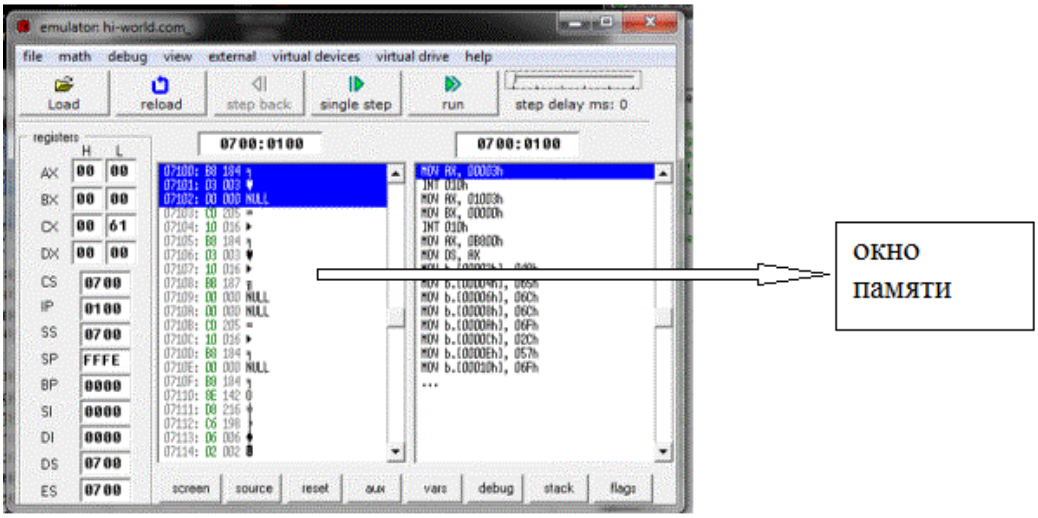
#MAKE_BIN# - простой исполнимый файл.

#MAKE_BOOT# - эта директива копирует первую дорожку дискеты (загрузочный сектор).

3. Выберите "examples"- (Hello,world) из меню "File".

4. Щелкните кнопку [emulate] (или нажмите клавишу F5).

5. Щелкните кнопку [Single Step] (пошаговый режим) (или нажмите клавишу F8), и наблюдайте за выполнением кода.



В окне памяти: первая строка - смещение, вторая строка - значение hexadecimal, третья строка - десятичное значение, и последняя строка - значение символа ASCII.

Кнопка [Single Step] выполняет команды, один за другим останавливающие после каждой команды.

[Run] кнопка выполняет команды один за другим с задержкой, установленной задержкой шага между командами.

6. Дважды щелкните на текстовых полях регистра - открывается окно "Extended Viewer " со значением того регистра, преобразованного ко всем возможным формам. Вы можете изменять значение регистра непосредственно в этом окне.

7. Дважды щелкните на элементе списка памяти - открывается " Extended Viewer" со значением WORD, загруженным со списка памяти в выбранном местоположении.

Менее существенный байт - в младшем адресе: LOW BYTE загружен от выбранной позиции и HIGH BYTE от следующего адреса памяти. Можно изменять значение слова памяти непосредственно в окне "Extended Viewer", можно изменять значения регистров во времени выполнения, печатая по существующим значениям

Задание 2. Разработка программы типа com на языке ассемблер

1. Щелкните на кнопку New, выберите com и введите в строке «add your code here» следующие команды:

№ строки	Команда
1	org 100h
2	begin:
3	mov ah, 9
4	mov dx,offset message
5	int 21h
6	ret
7	message db "Hello", 0dh, 0ah, '$'
8	end begin

Задание 3. Создание программы типа *. EXE с использованием простых арифметических действий

Простые арифметические операторы.

Сложение.

Команда ADD (Addition - сложение (гл. to add - сложить)) осуществляет сложение первого и второго операндов. Исходное значение первого операнда (приемника) теряется, замещаясь результатом сложения. Второй операнд не изменяется. В качестве первого операнда команды ADD можно указывать регистр (кроме сегментного) или ячейку памяти, в качестве второго - регистр (кроме сегментного), ячейку памяти или

непосредственное значение, однако не допускается определять оба операнда одновременно как ячейки памяти. Операнды могут быть байтами или словами и представлять числа со знаком или без знака. Команду ADD можно использовать для сложения как обычных целых чисел, так и двоично-десятичных (с использованием регистра AX для хранения результата). Команда воздействует на флаги OF, SF, ZF, AF, PF и CF.

Команда	Назначение	Процессор
ADD приемник, источник	Сложение	8086

Щелкните на кнопку New, выберите Exe и введите в строке «add your code here» следующие команды для расчета выражения:

((5 + (AL – AH)) – BH) + BL где AL=9, BL=3, AH=4, BH=6.

: (№ + AL) – (BH + AH) – BL

```
mov al, 9
mov ah, 4
sub al, ah
mov dl, 5
add al, dl
mov bh, 6
sub al, bh
mov bl, 3
add al, bl
```

Для вывода результирующего значения в 10-ричной системе счисления добавьте следующий код:

```
mov bx, 0
mov bl, al
 mov ax, bx
push -1
mov cx,10
1:mov dx,0
div cx
push dx
cmp ax,0
jne 1
mov ah,2h
12:pop dx
cmp dx,-1
je ex
add dl,'0'
int 21h
jmp 12
ex:mov ax,4c00h
int 21h
end start
```

Контрольные задания:

Произвести расчет выражений:

1) (№ + AL) – (BH + AH) – BL
2) ((BH + (№ – AH)) – BL) + AL

где № - порядковый номер по журналу, AL=9, BL=3, AH=4, BH=6.

Контрольные вопросы:

1. Характеристика структуры файла типа *.com?
2. Какова структура ассемблерной программы?
3. С какой целью в код программы на ассемблере для DOS вводится строка ORG 100h?
4. Назначение команды MOV?
5. Прерывания 21h и 20h. Назначение?
6. Сущность и целесообразность использования команды RET вместо прерывания 20h?

Лабораторная работа №6

Тема: Программирование на языке Ассемблер в среде операционной системы MS DOS

Цель работы: *приобретение навыков программирования на языке Ассемблер в среде операционной системы MS DOS.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Простые арифметические операторы.

Арифметические команды любого микропроцессора привлекают к себе наибольшее внимание. Каждый заинтересован в выполнении арифметических вычислений, и именно эти команды проделывают такую работу. Хотя их немного, они выполняют большинство преобразований данных, а микропроцессоре. В реальных же условиях арифметические команды занимают лишь малую часть всех исполняемых команд.

Сложение.

Команда ADD (Addition - сложение (гл. to add - сложить)) осуществляет сложение первого и второго операндов. Исходное значение первого операнда (приемника) теряется, замещаясь результатом сложения. Второй операнд не изменяется. В качестве первого операнда команды ADD можно указывать регистр (кроме сегментного) или ячейку памяти, в качестве второго - регистр (кроме сегментного), ячейку памяти или непосредственное значение, однако не допускается определять оба операнда одновременно как ячейки памяти. Операнды могут быть байтами или словами и представлять числа со знаком или без знака. Команду ADD можно использовать для сложения как обычных целых чисел, так и двоично-десятичных (с использованием регистра AX для хранения результата). Команда воздействует на флаги OF, SF, ZF, AF, PF и CF.

Примеры:

mov al,10 - --> загружаем в регистр AL число 10
add al,15 - --> al = 25; al - приемник, 15 - источник
mov ax,25000 - --> загружаем в регистр AX число 25000
add ax,10000 - --> ax = 35000; ax - приемник, 10000 - источник
mov cx, 200 - --> загружаем в регистр CX число 200
mov bx,760 - --> a в регистр BX - 760
add cx,bx - --> cx = 960, bx = 760 (bx не меняется); cx - приемник, bx - источник

Задание.

1. Запустить эмулятор EMU8086.

2. Получите задание у преподавателя (один из пяти вариантов табл. №1) и, пользуясь правилами оформления ассемблерных программ, напишите программы расчета значения A (два-три варианта).

3. Программу ассемблируйте в файл типа *. Exe.

№ варианта	Расчетная формула	B	C	D
1	A=B+C-D	1	35	23
2	A=B+C+D	65	1	1
3	A=C-D+B	1	33	1
4	A=D+A-B	18	1	88
5	A= B-C+D	45	10	1

Контрольные вопросы:

1. Структура файлов типа *. exe?
2. Структурные отличия файлов *. exe от *.com в операционной среде DOS?
3. Команда add основное назначение?
4. Команда sub основное назначение?
5. Команда inc основное назначение?
6. Команда dec основное назначение?

Лабораторная работа №7

Тема: Программирование на языке Ассемблер в среде операционной системы MS DOS

Цель работы: *изучить арифметические операции языка ассемблера; научиться их использовать при составлении программ; научиться применять битовые команды*.

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Арифметические операции на языке ассемблера выполняются над целыми числами четырех типов:

Беззнаковыми двоичными, знаковыми двоичными, упакованными десятичными и неупакованными десятичными.

В данной лабораторной работе рассматриваются арифметические операции над беззнаковыми числами.

Используются следующие команды:

ADD – сложить, SUB – вычесть.

Описание команд: Работают с 8 и 16 битовыми операндами, инструкция ADD выполняет сложение операнда источника (правого операнда) с содержимым операнда приемника (левый операнд), результат помещается в операнд приемник. Инструкция SUB делает тоже самое, только она вычитает операнд источник из операнда приемника, результат помещается в операнд приемник. Операндами могут быть регистры, константы, ячейки памяти в различных комбинациях, но нельзя добавить (вычесть) значение одной ячейки памяти к другой, а также в качестве операнда источника использовать константу (непосредственное значение). Команда воздействует на шесть флагов: AF, CF, OF, PF, SF, ZF.

Например, флаг переноса CF=1 если результат действия не помещается в операнде приемнике, в противном случае CF=0.

MUL - умножить.

Описание команд: инструкция MUL перемножает 8 и 16 битовые беззнаковые множители, создавая 16 или 32 битовое произведение. При 8 битовом произведении один из операндов в регистре AL другой может быть 8 битовым регистром общего назначения или переменной памяти соответствующего назначения. Результат помещается в регистр AX (16 битовый). При 16 битовых множителях один из сомножителей в 16 битовом регистре общего назначения другой в переменной памяти, 32 битовый результат в регистрах DX:AX. При этом младшие 16 бит в AX старшие в DX.

Команда воздействует на два флага: CF, OF.

DIV – разделить.

Описание команд: позволяет разделить 32 битовое значение на 16 битовое значение или 16 битовое на 8 битовое. При делении 16 битового значения делимое помещается в AX, 8 битовый делитель помещается в регистр или в переменную соответствующего размера. Результат (8 битовый) помещается в AL, остаток в AH.

Состояние флагов не определено, но если частное не помещается в регистре AL (AX) процессор генерирует прерывание типа 0 (деление на 0). В заданиях используются директивы и команды, изученные в предыдущих лабораторных работах.

БИТОВЫЕ КОМАНДЫ

Битовые команды рассматривают свои операнды не в виде привычных уже байтов, слов и двойных слов, в виде последовательности битов. Эти команды реализуют логические операции и команды сдвигов.

Логические операции (или булевы команды) – как, следует из названия, выполняют логические операции – отрицание, конъюнкцию, дизъюнкцию и им присуще ряд черт.

Инвертировать	`not opr`
"И" (конъюнкция)	`and dst,src`
Логическое сравнение	`test opr1,opr2`
"ИЛИ" (дизъюнкция)	`or dst,src`
"Исключающее ИЛИ"	`xor dst,src`

Примечание: команда sal при трансляции будет воспринята как shl, так кА это разные мнемонические названия одной и той машинной команды.

Циклические сдвиги. Особенность циклических сдвигов в том, что «уходящий» бит не теряется, а возвращается в операнд, но с другого конца.

```
Циклический сдвиг влево (shift arithmetic left):      ROL
Циклический сдвиг вправо (shift arithmetic right):    ROR
   Например:
   mov   ah,11000011b
   rol   ah,1              ;CF=1, al=10000111b
   mov   ah,11100010b
   ror   ah,1              ;CF=0, al=01110001b
```

Задание. Задача заключается в вычислении результата выполнения арифметического выражения, в котором некоторые числа постоянны, а другие переменные.

Формула вычислений: X=(A*2+B*C)/(D-3)

Приведенная программа сначала резервирует ячейки памяти под переменные, затем выполняет умножение однобайтовых чисел (A*2), результат умножения – двухбайтовое число в регистре AX, сохраняется в регистре CX, далее выполняется умножение однобайтовых чисел (B*C), результат – двухбайтовое число храниться в аккумуляторе AX. После сложения двух сомножителей и вычисления знаменателя (B-3) выполняется деление. Результат присваивается переменной X.

1. Наберите приведенную программу 1, запишите исходный файл с расширением *.asm, получите файл с расширением *.exe.
2. Выполните программу с 5 вариантами различных начальных значений переменных A, B, C, D по шагам (см. таблицу 1) и запишите результат выполнения в таблицу (в регистре AL – частное, AH – остаток). Переведите результат в десятичную систему.

Таблица 1

Вариант		1	2	3	4	5
Значения	A	3	0AH	20	60	20H
	B	4	5H	4	16	9H
	C	2	8H	15	5	4H
	D	5	9H	6	18	1CH
Частное AL						
Остаток AH						

```
;программа 1
;x =(a*2+b*c)/(d-3)
.model small
.stack 100h
.data
a   db    ?
b   db    ?
c   db    ?              ;Резервируем память для переменных
d   db    ?              ; A,B,C,D,X
x   dw    ?
.code
start:
mov   ax,@data
mov   ds,ax
mov   a,3
mov   b,4
mov   c,2
mov   d,5
mov   al,2
mul   a
mov   cx,ax
mov   al,b
      mul   c
      add   ax,cx
      mov   cl,d
      sub   cl,3
      div   cl
      mov   x,ax
      mov   ah,4ch
      int   21h
      end   start
```

3. Наберите приведенную программу 2, запишите исходный файл с расширением *.asm, получите файл с расширением *.exe.
4. Выполните программу с 5 вариантами различных начальных значений регистра al. Переведите результат и запишите таблицу 2 в двоичной и десятичной системе.

```
;программа 2
.model tiny
.stack 100h
.code
start:
mov ah, 01001101b
shr ah,1
mov ah, 01001101b
shl ah,1
mov ah, 01001101b
sar ah,1
mov ah, 01001101b
ror ah,1
mov ah, 01001101b
rol ah,1
mov ax,4c00h
int 21h
end start
```

Контрольные вопросы:

1. Чем отличается выполнение арифметических операций на языке ассемблера от языков высокого уровня?
2. По какому биту регистра флагов можно установить, что предшествующее вычитание привело к отрицательному результату?
3. По какому биту регистра флагов можно установить, что результат арифметической операции превысил разрядную сетку?
4. В чем особенность выполнения арифметических операций с знаковыми и беззнаковыми числами?
5. В каком регистре необходимо указывать величину сдвига в команде сдвига?

Лабораторная работа №8

Тема: Поддержка ввода с клавиатуры и вывода на экран средствами BIOS и DOS

Цель работы: *способствовать формированию навыков программирования ввода с клавиатуры и вывода на экран средствами BIOS и DOS на языке Ассемблер.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

1. Средства DOS.

Функция DOS 02h - Записать символ в STDOUT с проверкой на Ctrl-Break.

Ввод:	AH = 02h DL = ASCII-код символа

Вывод:	Никакого, согласно документации, но на самом деле: AL = код последнего записанного символа (равен DL, кроме случая, когда DL = 09h (табуляция), тогда в AL возвращается 20h).

Пример № 1

```
org 100h                          ; начало COM-файла
begin:                            ; метка начала кода программы
mov dl,< ASCII-код символа >      ; заносим в регистр dl - любой ASCII-код символа
mov ah,2                          ; номер функции DOS "вывод символа"
int 21h                           ; вызов DOS
ret                               ; функция DOS "завершить программу"
end begin                         ; метка окончания кода программы
```

Эта программа, выводит на экран любой ASCII-символ, в установленную позицию курсора.

Все функции DOS вывода на экран используют устройство STDOUT, стандартный вывод. Это позволяет перенаправлять вывод программы в файл или на стандартный ввод другой программы. Например, если откомпилировать приведен пример (создать файл cod.com) и написать в командной строке

cod.com > cod. Out то на экран ничего выдано не будет, а в текущем каталоге появится файл cod. out, содержащий ASCII-код символа.

Функция DOS 06h - Записать символ в STDOUT без проверки на Ctrl-Break

Ввод:	AH = 06h DL = ASCII-код символа (кроме FFh)
Вывод:	Никакого, согласно документации, но на самом деле: AL = код записанного символа (копия DL)

Эта функция не обрабатывает управляющие символы (CR, LF, HT и BS выполняют свои функции при выводе на экран, но сохраняются при перенаправлении вывода в файл) и не проверяет нажатие Ctrl-Break.

Заменим в примере №1 MOV AH,2 на MOV AH,6 и перекомпилируем этот пример. Работу откомпилированного примера смотрим в операционной системе MS-DOS.

Функция DOS 09h - Записать строку в STDOUT с проверкой на Ctrl-Break.

Ввод:	AH = 09h DS: DX = адрес строки, заканчивающейся символом $ (24h)
Вывод:	Никакого, согласно документации, но на самом деле: AL = 24h (код последнего символа)

Действие этой функции полностью аналогично действию функции 02h, но выводится не один символ, а целая строка (практическая работа 1).

Функция DOS 40h - Записать в файл или устройство

Ввод:	AH = 40h BX = 1 для STDOUT или 2 для STDERR DS: DX = адрес начала строки CX = длина строки
Вывод:	CF = 0, AX = число записанных байт

Эта функция предназначена для записи в файл, но, если в регистр BX поместить число 1, функция 40h будет выводить данные на STDOUT, а если BX = 2 - на устройство STDERR. STDERR всегда выводит данные на экран и не перенаправляется в файлы. На этой функции основаны используемые в C функции стандартного вывода - фактически функция C fputs () просто вызывает это прерывание, помещая свой первый аргумент в BX, адрес строки (второй аргумент) - в DS: DX и длину - в CX.

Пример № 2

org 100h	; начало COM-файла
begin:	; метка начала кода программы
mov ah,40h	; номер функции DOS
mov bx,2	; указываем устройство STDERR
mov dx,offset message	; DS: DX - адрес строки
mov cx,25	; CX - длина строки
int 21h	; вызов DOS
ret	; функция DOS "завершить программу"
message db "This function can print $"	; строка содержащая выводимые данные.
end begin	; метка окончания кода программы

Если скомпилировать этот пример и запустить ее командой
dosout.com > dosout. out
то сообщение появится на экране, а файл dosout2. out окажется пустым.
Прерывание INT 29H: Быстрый вывод символа на экран

Ввод:	AL = ASCII-код символа

Задание.

1. Запустите эмулятор EMU8086.2. Напишите программы примеры, которых приведены в данной лабораторной работе, каждый раз выбирая тип исполняемого файла .com (**ASCII-коды символов в приложении**).

3. Изучите структуру откомпилированных программ.

4. Получите задание у преподавателя (один из пяти вариантов табл. №1) напишите программу вывода на экран строки ‘Hello’.

5. Напишите программу работы переключения SuperVGA-видеорежимов (согласно варианту табл. №2)

Таблица 1.

№ вар.	Функция вывода (DOS)	Функция вывода (BIOS)	Видеопамять
1	02h	Ah=02h	‘Hello’
2	06h	Ah=08h	
3	09h	Ah=09h	
4	40h	Ah=0Ah	
5	29h	Ah=13h	

Примечание: В примерах, в которых возможно задание различных параметров вывода (цвет символа, фона; номер строки, столбца, страницы и т.д.) выводите на экран слово "hello" с параметрами отличными от стандартных.

Таблица 2.

№ Варианта	SuperVGA-видеорежим
1	108h
2	109h
3	10Ah
4	108h
5	10Ch

Контрольные вопросы:

1. Перечислите функции вывода на экран средствами операционной системы DOS?
2. Принцип работы функции DOS 02h?
3. Укажите основные управляющие символы вывода на экран?
4. Укажите отличие функции DOS 02h от 06h?

5. С помощью, каких функций можно установить нужный видеорежим (текстовый, цветной, монохромный)?

Лабораторная работа №9

Тема: Программирование целочисленных вычислений

Цель работы: *способствовать формированию навыков программирования арифметических операций: умножения и деления на языке Ассемблер.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Команда MUL может быть записана в трех различных форматах — в зависимости от операнда:

```
MUL r/m8
MUL r/m16
MUL r/m32
```

В 8-разрядной форме операнд может быть любым 8-битным регистром или адресом памяти. Второй операнд всегда хранится в AL. Результат (произведение) будет записан в регистр AX.

```
(r/m8) * AL -> AX
```

В 16-разрядной форме операнд может быть любым 16-битным регистром или адресом памяти. Второй операнд всегда хранится в AX. Результат сохраняется в паре DX:AX.

```
(r/m16) * AX -> DX:AX
```

В 32-разрядной форме второй операнд находится в регистре EAX, а результат записывается в пару EDX:EAX.

```
(r/m32) * EAX -> EDX:EAX
```

Пример 1: умножить значения, сохраненные в регистрах BH и CL, результат сохранить в регистр AX:

```
mov al, bh    ;AL = BH – сначала заносим в AL второй операнд
mul cl        ;AX = AL * CL – умножаем его на CL
```

Результат будет сохранен в регистре AX.

Пример 2: вычислить 4862 результат сохранить в DX:AX:

```
mov ax, 486   ; AX = 486
mul ax        ; AX * AX -> DX:AX
```

Команды DIV и IDIV

Команда DIV может быть представлена в трех различных форматах в зависимости от типа операнда:

```
DIV r/m8
DIV r/m16
DIV r/m32
```

Операнд служит делителем, а делимое находится в фиксированном месте (как в случае с MUL).

В 8-битной форме переменный операнд (делитель) может быть любым 8-битным регистром или адресом памяти. Делимое содержится в AX. Результат сохраняется так: частное — в AL, остаток —- в AH.

$$AX / (r/m8) \rightarrow AL, \text{остаток} \rightarrow AH$$

В 16-битной форме операнд может быть любым 16-битным регистром или адресом памяти. Второй операнд всегда находится в паре DX:AX. Результат сохраняется в паре DX:AX (DX — остаток, AX — частное).

$$DX:AX / (r/m16) \rightarrow AX, \text{остаток} \rightarrow DX$$

В 32-разрядной форме делимое находится в паре EDX:EAX, а результат записывается в пару EDX:EAX (частное в EAX, остаток в EDX).

$$EDX:EAX / (r/m32) \rightarrow EAX, \text{остаток} \rightarrow EDX$$

Команда IDIV используется для деления чисел со знаком, синтаксис ее такой же, как у команды DIV.

Пример 3: разделить 13 на 2, частное сохранить в BL, а остаток в — BH:

```
mov ax,13       ; AX = 13
mov cl,2        ; CL = 2
div cl          ; делим на CL
mov bx,ax       ; ожидаемый результат находится в AX,
                ; копируем в BX
```

Задание.

1. Запустите эмулятор EMU8086.

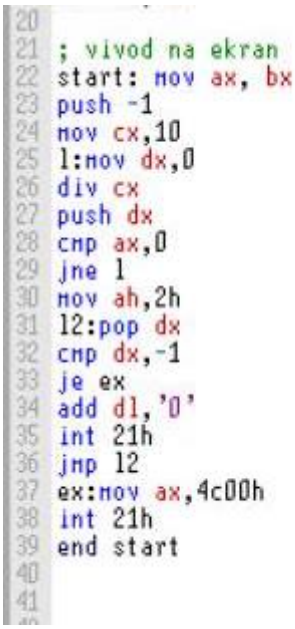
2. Напишите программы примеры, которых приведены в данной лабораторной работе, каждый раз выбирая тип исполняемого файла .com.

3. Ввести в эмулятор программу вычисляющую следующее выражение:

(AL * 40) / (BH * AH) где AL=2,BL=3,AH=4,BH=5.

4. В пошаговом режиме проверить регистры. Записать программу в отчет, дополнив комментарии к каждой строке.

```
05 org 100h
06
07 xor ax, ax
08 mov al, 2
09 mov dl, 40
10 mul dl
11 mov dl, al
12 mov bh, 5
13 mov al, 4
14 mul bh
15 mov dh, al
16 mov al, dl
17 div dh
18 mov bx, 0
19 mov bx, ax
```



```
; vivod na ekran
start: mov ax, bx
push -1
mov cx,10
1:mov dx,0
div cx
push dx
cmp ax,0
jne 1
mov ah,2h
12:pop dx
cmp dx,-1
je ex
add dl,'0'
int 21h
jmp 12
ex:mov ax,4c00h
int 21h
end start
```

5. Произвести расчет значений:
    1. (№ * AL) / BL
    2. (BH * (AH / №) / BL)

где № - порядковый номер по журналу, AL=15, BL=4, AH=30, BH=4.

Контрольные вопросы:

1. Для чего используется команда **MUL?**
2. В каких форматах может быть записана команда **MUL**?
3. Для чего используется команда **MUL?**
4. Где содержится делимое и результат в 8-битной форме?

Лабораторная работа №10

Тема: Программирование логических вычислений

Цель работы: *закрепить навыки программирования логических операций на языке Ассемблер.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Логические операции являются важным элементом в проектировании микросхем и имеют много общего в логике программирования. Команды AND, OR, XOR и TEST - являются командами логических операций. Эти команды используются для сброса и установки бит и для арифметических операций в коде ASCII. Все эти команды обрабатывают один байт или одно слово в регистре или в памяти, и устанавливают флаги CF, OF, PF, SF, ZF.

Команда AND

Команда AND (Логическое И) осуществляет логическое (побитовое) умножение первого операнда на второй. Исходное значение первого операнда (приемника) теряется, замещаясь результатом умножения. В качестве первого операнда команды and можно указывать регистр (кроме сегментного) или ячейку памяти, в качестве второго - регистр (кроме сегментного), ячейку памяти или непосредственное значение, однако не допускается определять оба операнда одновременно как ячейки памяти. Операнды могут быть байтами или словами. Команда воздействует на флаги SF, ZF и PF.

Правила побитового умножения:

Первый операнд-бит 0101	Бит результата 0001
Второй операнд-бит 0011	

Пример 1

mov AX,0FFEh
and AX,5555h; AX=0554h

Пример 2

mov ax,00101001b
add ax,11110111b ; ax=00100001b

Команда OR

Команда OR (Логическое ВКЛЮЧАЮЩЕЕ ИЛИ) выполняет операцию логического (побитового) сложения двух операндов. Результат замещает первый операнд (приемник); второй операнд (источник) не изменяется. В качестве первого операнда можно указывать регистр (кроме сегментного) или ячейку памяти, в качестве второго - регистр (кроме сегментного), ячейку памяти или непосредственное значение, однако не допускается определять оба операнда одновременно как ячейки памяти. Операнды команды OR могут быть байтами или словами. Команда воздействует на флаги OF, SF, ZF, PF и CF, при этом флаги CF и OF всегда сбрасываются в 0.

Правила побитового сложения:

Первый операнд-бит 0101	Бит результата 0011
Второй операнд-бит 0111	

Пример 1

mov AX,000Fh
mov BX,00F0h
or AX,BX; AX=00FFh, BX=00F0h

Пример 2

mov AX,00101001b
mov BX,11110111b
or AX,BX ; mov dx,11111111b

Пример 3

mov AX,000Fh
or AX,8001h ; AX=800Fh

Команда XOR

Команда XOR (Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ) выполняет операцию логического (побитового) ИСКЛЮЧАЮЩЕГО ИЛИ над своими двумя операндами. Результат операции замещает первый операнд; второй операнд не изменяется. Каждый бит результата устанавливается в 1, если соответствующие биты операндов различны, и сбрасывается в 0, если соответствующие биты операндов совпадают.

В качестве первого операнда команды XOR можно указывать регистр (кроме сегментного) или ячейку памяти, в качестве второго - регистр (кроме сегментного), ячейку памяти или непосредственное значение, однако не допускается определять оба операнда одновременно как ячейки памяти. Операнды могут быть байтами или словами. Команда

воздействует на флаги OF, SF, ZF, PF и CF, причем флаги OF и CF всегда сбрасываются, а остальные флаги устанавливаются в зависимости от результата.

Правила побитового исключающего или:

Первый операнд-бит 0101	Бит результата 0011
Второй операнд-бит 0110	

Команда TEST

Команда TEST (Логическое сравнение) выполняет операцию логического умножения И над двумя операндами и, в зависимости от результата, устанавливает флаги SF, ZF и PF. Флаги OF и CF сбрасываются, а AF имеет неопределенное значение. Состояние флагов можно затем проанализировать командами условных переходов. Команда TEST не изменяет ни один из операндов.

Команда NOT

Команда NOT (NOT Инверсия, дополнение до 1, логическое отрицание) выполняет инверсию битов указанного операнда, заменяя 0 на 1 и наоборот.

Задание.

1. Выполнить все примеры на эмуляторе EMU8086.
2. Набрать и проверить работоспособность программы производящую работу с логическими операторами.

```
#make_com#

org 100h

mov al,1        ;запись в AL=1
mov bl,1        ;запись в BL=1

and al,bl       ;AL и BL = 0

ret
```

```
#make_com#

org 100h

mov al,1        ;запись в AL=1
mov bl,0        ;запись в BL=1

or al,bl       ;AL и BL = 1

ret
```

```
#make_com#

org 100h

mov al,2        ;запись в AL=2
mov bl,al       ;запись в BL=2

not al      ;AL = -3
neg bl      ;BL = -2

ret
```

3. Произвести расчет значений:

1. 5 + AL и BH
2. AL или BH - 5
3. 10 – CL не AL

где BH - порядковый номер по журналу, AL= 1, CL=3, DL=5

Контрольные вопросы:

1. Назначение команды AND?
2. На какие флаги воздействует команда AND?
3. Назначение команды OR?
4. На какие флаги воздействует команда OR?
5. Назначение команды XOR?
6. Назначение команды TEST?
7. Назначение команды NOT?

Лабораторная работа №11

Тема: Программирование ветвлений и итерационных циклов

Цель работы: *изучить команды безусловного перехода языка ассемблера; команды условного перехода; научиться организовывать циклы на ассемблере.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Лишь в самых простых программах команды выполняются в порядке их размещения в памяти. Изменение естественного порядка выполнения команд обеспечивается с помощью команд передачи управления, весь набор которых можно разделить на 3 группы: команды безусловной передачи управления, команды условной передачи управления, команды управления циклами.

К первой группе относятся команды безусловных переходов и команды вызова процедур (безусловных переходов с возрастом). Команды вызова процедур будут рассмотрены в следующей работе.

Безусловные переходы

Основной инструкцией перехода в наборе инструкций процессора 8086 является инструкция JMP. Эта инструкция указывает процессору, что в качестве следующей за JMP инструкцией нужно выполнить инструкцию по целевой метке. Например, после завершения выполнения фрагмента программы:

```
        mov ax,1
        jmp metka
        add ax,5
metka:
        inc ax
```

Регистр AX будет содержать значение 2, а инструкция ADD, следующая за меткой metka не будет выполнена. Здесь инструкция jmp указывает процессору, что нужно установить указатель инструкций IP в значение смещения метки metka, поэтому следующей выполняемой инструкцией будет инструкция inc ax.

Инструкцию JMP можно использовать для перехода в другой сегмент кода, загружая в одной инструкции и регистр CS, и регистр IP. Если вы хотите, чтобы метка принудительно интерпретировалась, как метка дальнего типа, можно использовать операцию FAR PTR.

Наконец, вы можете выполнить переход по адресу, записанному в регистре или в переменной памяти. Например:

```
mov ax,OFFSET TestLabel
jmp ax
...
TestLabel:
```

Условные переходы

Описанные в предыдущем разделе инструкции переходов - это только часть того, что вам потребуется для написания полезных программ. В действительности необходима возможность писать такие программы, которые могут принимать решения. Именно это можно делать с помощью операций условных переходов.

Набор инструкций процессора 8086 предусматривает большое разнообразие инструкций условных переходов, что позволяет вам осуществлять переход почти по любому флагу или их комбинации. Можно осуществлять условный переход по состоянию нуля, переноса, по знаку, четности или флагу переполнения и по комбинации флагов, показывающих результаты операций чисел со знаками.

Перечень инструкций условных переходов приводится в таблице 1.

Таблица 1 - **Инструкции условных переходов**

Название	**Значение**	**Проверяемые флаги**
JB/JNAE	Перейти, если меньше / перейти, если не больше или равно	CF = 1
JAE/JNB	Перейти, если больше или равно / перейти, если не меньше	CF = 0
JBE/JNA	Перейти, если меньше или равно / перейти, если не больше	CF = 1 или ZF = 1
JA/JNBE	Перейти, если больше / перейти, если не меньше или равно	CF = 0 и ZF = 0
JE/JZ	Перейти, если равно	ZF = 1
JNE/JNZ	Перейти, если не равно	ZF = 0
JL/JNGE	Перейти, если меньше чем / перейти, если не больше чем или равно	SF = OF
JGE/JNL	Перейти, если больше чем или равно /перейти, если не меньше чем	SF = OF
JLE/JNLE	Перейти, если меньше чем или равно / перейти, если не больше, чем	ZF = 1 или SF = OF
JG/JNLE	Перейти, если больше чем / перейти, если не меньше чем или равно	ZF = 0 или SF = OF
JP/JPE	Перейти по четности	PF = 1
JNP/JPO	Перейти по нечетности	PF = 0
JS	Перейти по знаку	SF = 1
JNS	Перейти, если знак не установлен	SF = 0
JC	Перейти при наличии переноса	CF = 1
JNC	Перейти при отсутствии переноса	CF = 0
JO	Перейти по переполнению	OF = 1
JNO	Перейти при отсутствии переполнения	OF = 0

CF – флаг переноса,
SF – флаг знака,

OF – флаг переполнения,
ZF – флаг нуля,
PF – флаг четности.
Несмотря на свою гибкость, инструкции условного перехода имеют также серьезные ограничения, поскольку переходы в них всегда короткие. Другими словами, целевая метка, указанная в инструкции условного перехода, должна отстоять от инструкции перехода не более, чем на 128 байт.

Задание В соответствии с вариантом напишите программу на языке ассемблера с полным описанием сегментов для вычисления значения y.

1	$y = y1 + y2;\ y1 = \begin{cases} a + x, & \text{если } x > a \\ 2a - x, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a * x, & \text{если } x > 10 \\ x, & \text{если } x <= 10 \end{cases}.$
2	$y = y1 - y2;\ y1 = \begin{cases} x - 2, & \text{если } x >= 2 \\ 8, & \text{если } x < 2 \end{cases};\ y2 = \begin{cases} 4, & \text{если } x = 0 \\ a - x, & \text{если } x <> 0 \end{cases}.$
3	$y = y1 * y2;\ y1 = \begin{cases} x - a, & \text{если } x > a \\ 5, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a, & \text{если a} > x \\ a * x, & \text{если a} <= x \end{cases}.$
4	$y = y1 + y2;\ y1 = \begin{cases} 2 - x, & \text{если } x < 2 \\ a + 3, & \text{если } x >= 2 \end{cases};\ y2 = \begin{cases} a - 1, & \text{если } x < a \\ a * x - 1, & \text{если } x >= a \end{cases}.$
5	$y = y1 - y2;\ y1 = \begin{cases} \lvert x \rvert, & \text{если } x < 0 \\ x - a, & \text{если } x >= 0 \end{cases};\ y2 = \begin{cases} a + x, & \text{если } x \bmod 3 = 1 \\ 7, & \text{в остальных случаях} \end{cases}.$
6	$y = y1 + y2;\ y1 = \begin{cases} x \bmod 4, & \text{если } x > a \\ a, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a * x, & \text{если } x / a > 3 \\ x, & \text{если } x / a <= 3 \end{cases}.$
7	$y = y1 + y2;\ y1 = \begin{cases} 4 - x, & \text{если } \lvert x \rvert < 3 \\ a + x, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} 2, & \text{если } x \text{ четное} \\ a + 2, & \text{в остальных случаях} \end{cases}.$
8	$y = y1 + y2;\ y1 = \begin{cases} 4 * x, & \text{если } x <= 4 \\ x - a, & \text{если } x > 4 \end{cases};\ y2 = \begin{cases} 7, & \text{если } x \text{ нечетное} \\ x / 2 + a, & \text{в остальных случаях} \end{cases}.$
9	$y = y1 * y2;\ y1 = \begin{cases} a * x, & \text{если } x \bmod 3 = 2 \\ 9, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} a - x, & \text{если } a > x \\ a + 2, & \text{если } a <= x \end{cases}.$
10	$y = y1 - y2;\ y1 = \begin{cases} a + \lvert x \rvert, & \text{если } x > a \\ a - 7, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a * 3, & \text{если } a > 3 \\ 11, & \text{если a} <= 3 \end{cases}.$
11	$y = y1 \bmod y2;\ y1 = \begin{cases} 10 + x, & \text{если } x > 1 \\ \lvert x \rvert + a, & \text{если } x <= 1 \end{cases};\ y2 = \begin{cases} 2, & \text{если } x > 4 \\ x, & \text{если } x <= 4 \end{cases}.$
12	$y = y1 / y2;\ y1 = \begin{cases} 15 + x, & \text{если } x > 7 \\ \lvert a \rvert + 9, & \text{если } x <= -7 \end{cases};\ y2 = \begin{cases} 3, & \text{если x} > 2 \\ \lvert x \rvert - 5, & \text{если } x <= 2 \end{cases}.$
13	$y = y1 * y2;\ y1 = \begin{cases} 3 + x, & \text{если } x = a \\ a - x, & \text{если } x <> a \end{cases};\ y2 = \begin{cases} \lvert a \rvert, & \text{если } a < x \\ \lvert a \rvert - x, & \text{если ```a} >= x \end{cases}.$
14	$y = y1 - y2;\ y1 = \begin{cases} 2 * x + a, & \text{если } x > 2 \\ 2 * x + 1, & \text{если } x <= 2 \end{cases};\ y2 = \begin{cases} \lvert x \rvert + 1, & \text{если x} > 0 \\ a - 1, & \text{если } x <= 0 \end{cases}.$
15	$y = y1 \bmod y2;\ y1 = \begin{cases} 8 + \lvert x \rvert, & \text{если } x < 1 \\ \lvert a \rvert * 2, & \text{если } x >= 1 \end{cases};\ y2 = \begin{cases} 3, & \text{если } x = a \\ a + 1, & \text{если } x <> a \end{cases}.$
16	$y = y1 + y2;\ y1 = \begin{cases} 4 + x, & \text{если } x <= 3 \\ a * x, & \text{если } x > 3 \end{cases};\ y2 = \begin{cases} \lvert a \rvert - 2, & \text{если x} > a \\ x, & \text{если } x <= a \end{cases}.$

17	$y = y1 - y2;\ y1 = \begin{cases} a + \lvert x\rvert, & \text{если } x < 0 \\ x - a, & \text{если } x >= 0 \end{cases};\ y2 = \begin{cases} 7, & \text{если } x < 3 \\ a, & \text{если } x >= 3 \end{cases}.$
18	$y = y1 \bmod y2;\ y1 = \begin{cases} 7 + x, & \text{если } x < 3 \\ \lvert a\rvert + x, & \text{если } x >= 3 \end{cases};\ y2 = \begin{cases} 1, & \text{если } x > 5 \\ a + x, & \text{если } x <= 5 \end{cases}.$
19	$y = \lvert y1\rvert + y2;\ y1 = \begin{cases} -5, & \text{если } x > 4 \\ x - a, & \text{если } x <= 4 \end{cases};\ y2 = \begin{cases} \lvert a\rvert, & \text{если } x > a \\ 9, & \text{если } x <= a \end{cases}.$
20	$y = y1 * y2;\ y1 = \begin{cases} 2 * x, & \text{если } x < 5 \\ \lvert a\rvert + x, & \text{если } x >= 5 \end{cases};$
21	$y = y1 + \lvert y2\rvert;\ y1 = \begin{cases} 3, & \text{если } x \bmod 3 = 1 \\ x - a, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} a / x, & \text{если } x <> 0 \\ 4, & \text{если } x = 0 \end{cases}.$
22	$y = y1 - y2;\ y1 = \begin{cases} \lvert x\rvert + \lvert a\rvert, & \text{если } x <= 3 \\ x * a, & \text{если } x > 3 \end{cases};\ y2 = \begin{cases} 3, & \text{если } a = x \\ a - x, & \text{если } a <> x \end{cases}.$
23	$y = y1 + y2;\ y1 = \begin{cases} 2 * x, & \text{если } \lvert x\rvert > 4 \\ 4 + a, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} 9, & \text{если } x = 0 \\ a / x, & \text{если } x <> 0 \end{cases}.$
24	$y = y1 * y2;\ y1 = \begin{cases} x, & \text{если } x \bmod 4 <> 2 \\ a + x, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} a - x, & \text{если } x < a \\ a * x, & \text{если } x >= a \end{cases}.$
25	$y = y1 / y2;\ y1 = \begin{cases} 12, & \text{если } x < 12 \\ x + 1, & \text{если } x >= 12 \end{cases};\ y2 = \begin{cases} 2, & \text{если } x > 2 \\ a + x, & \text{если } x <= 2 \end{cases}.$

Контрольные вопросы:

1. С какими операциями используется команда JUMP?
2. Назовите сходство и различия команд перехода для знаковых и беззнаковых команд?
3. Принцип работы операторов цикла?
4. К какому типу переход (дальние, короткие) относятся условные переходы?
5. Как влияет команда CMP на регистр флагов?

Лабораторная работа №12

Тема: Программирование массивов и циклов

Цель работы: *способствовать формированию навыков программирования ветвлений и итерационных циклов на языке Ассемблер.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Циклы, позволяющие выполнить некоторый участок программы многократно, в любом языке являются одной из наиболее употребительных конструкций. В системе команд МП 86 циклы реализуются, главным образом, с помощью команды loop (петля), хотя имеются и другие способы организации циклов. В большинстве случаях число шагов в цикле определяется содержимым регистра CX, поэтому максимальное число шагов составляет 64 К.

В общем виде любой цикл записывается в ассемблере как условный переход.

Организация цикла с помощью команды LOOP (Первый способ).

Команда loop (анг. петля) выполняет декремент содержимого регистра CX (счетчик), и если оно не равно 0, осуществляет переход на указанную метку вперед или назад в том же программном сегменте в диапазоне - 128... + 127 байт. Обычно метка помещается перед первым предложением тела цикла, а команда loop является последней командой цикла. Содержимое регистра CX рассматривается как целое число без знака, поэтому максимальное число повторений группы включенных в цикл команд составляет 65536 (если перед входом в цикл CX=0). Команда не воздействует на флаги процессора.

Команда	Назначение	Процессор
LOOP метка	Организация циклов	8086

Пример 1. Организация циклического перехода (со счетчиком в регистре cx) на языке Assembler:

1	org 100h	; начало COM-файла
2	begin:	; метка начала кода программы
3	mov cx,10	; загружаем в (регистр-счетчик) CX количество повторов (отсчет будет идти от 10 до 0)
4	Label1:	; создаем метку (Label - метка).
5	mov ah,9	; помещаем номер функции DOS "вывод строки (9)" в регистр AH.
6	mov dx,offset String	; помещает в регистр DX смещение метки String относительно начала сегмента данных
7	int 21h	; функция DOS "вывод строки"
8	loop Label1	; оператор loop уменьшает на единицу CX и, если он не равен нулю, переходит на метку Label1 (строка 4)
9	ret	; функция DOS "завершить программу"
10	string db 'privet $'	; строка содержащая выводимые данные.
11	end begin	; метка окончания кода программы

В строке (3) загружаем в CX количество повторов (отсчет будет идти от 10 до 0). В строке (4) создаем метку (Label - метка). Далее (строки (5) - (8)) выводим сообщение. И в строке (8) оператор loop уменьшает на единицу CX и, если он не равен нулю, переходит на метку Label1 (строка (4)). Таким образом, строка будет выведена на экран десять раз. Когда программа перейдет на строку (9), регистр CX будет равен нулю.

Организация цикла с помощью команды JMP (Второй способ).

Команда jmp передает управление в указанную точку того же или другого программного сегмента. Адрес возврата не сохраняется. Команда не воздействует на флаги процессора.

Команда jmp имеет пять разновидностей:

- переход прямой короткий (в пределах - 128... + 127 байтов);
- переход прямой ближний (в пределах текущего программного сегмента);
- переход прямой дальний (в другой программный сегмент);
- переход косвенный ближний;
- переход косвенный дальний.

Пример 2.

1	org 100h	; начало COM-файла
2	begin:	; метка начала кода программы
3	label1:	; создаем метку
4	mov ah,9	; помещаем номер функции DOS "вывод строки (9)" в регистр AH.
5	mov dx,offset String	;помещает в регистр DX смещение метки String относительно начала сегмента данных
6	int 21h	; функция DOS "вывод строки"
7	jmp Label1	; переход на строку с меткой Label1
8	add cx,12	; прибавить к значению регистра cx число 12 (данная команда не выполняется)
9	dec cx	; уменьшить значение регистра cx на 1 (данная команда не выполняется)
10	ret	; функция DOS "завершить программу"
11	string db "PRIVET",13,10,'$'	; строка с содержащая выводимые данные.
12	end begin	; метка окончания кода программы

Задание. Ввести и проработать пример.

Задание 2. Ввести и проанализировать работу программу, выводящую на экран все ASCII-символы (16 строк по 16 символов в строке).

org 100h	; начало COM-файла
begin:	; метка начала кода программы
mov cx,256	; задаем значение счетчика (256 символов)
mov dl,0	; первый символ - с кодом 00
mov ah,2	; номер функции DOS "вывод символа"
cloop: int 21h	; вызов DOS
inc dl	; увеличение DL на 1 - следующий символ
test dl,0Fh	; если DL не кратен 16
jnz continue_loop;	; продолжить цикл,
push dx mov dl,0Dh	; иначе: сохранить текущий символ ; вывести CR
int 21h	; вызов DOS
mov dl,0Ah	; вывести LF
int 21h pop dx	; вызов DOS ; восстановить текущий символ
continue_loop:	; метка
loop cloop	; продолжить цикл
ret	; завершение COM-файла
end begin	; метка окончания кода программы

Здесь с помощью команды LOOP оформляется цикл, выполняющийся 256 раз (значение регистра CX в начале цикла). Регистр DL содержит код символа, который равен нулю в начале цикла и увеличивается каждый раз на 1 командой INC DL. Если значение DL сразу после увеличения на 1 кратно 16, оно временно сохраняется в стеке и на экран выводятся символы CR и LF, выполняющие переход на начало новой строки. Проверка выполняется командой TEST DL,0Fh - результат операции AND над DL и 0Fh будет нулем, только если младшие четыре бита DL равны нулю, что и соответствует кратности шестнадцати.

Задание 3

Набрать и проверить работоспособность программы с условными переходами (JNZ)

```
#make_com

org 100h

mov cx, 10
; создание метки: cx=cx-1
for_loop:dec cx

; проверка cx<>0, переход на метку
jnz for_loop

; если условие выполнилось, записать в регистр 17
mov ax, 17

ret
```

Набрать и проверить работоспособность программы с условными переходами (JE)

```
#make_com

org 100h

mov cx, 10

mov al, 1
mov dl, 1

; создание метки: cx=cx-1
for_loop:add al, dl

; проверка cx=0, переход на метку
je finish

; проверка cx<>0, переход на метку
loop for_loop

; если условие выполнилось, записать в регистр 17
finish:mov bl, 17

ret
```

Набрать и проверить работоспособность программы с безусловным переходом (JMP)

```
#make_com

org 100h

mov ax, 15

; создание метки: cx=cx-1
for_loop:dec ax

; проверка cx<>0, переход на метку
jmp for_loop

ret
```

Задание 4

1. Напишите программу, выводящую на экран слово "!!!!!!!!!! Hello!!!!!!!!!!" используя команды **циклических переходов** (3 варианта).
2. Напишите программу расчета суммы всех значений от 0 до 14.

Контрольные вопросы:

1. Что такое цикл?
2. Назначение команды loop?
3. Назначение команды jmp?
4. Назначение команд dec и jnz?

Лабораторная работа №13

Тема: Строковые функции

Цель работы: *изучить команды перемещения данных, команды повторения строковых инструкций и научиться организовывать массивы на ассемблере.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Строковые инструкции

Строковые инструкции отличаются от прочих инструкций процессора 8086. Они могут (в одной инструкции) обращаться к памяти и увеличивать или уменьшать регистр-указатель. Одна строковая инструкция может обращаться к памяти 130000 раз.

Строковые инструкции перемещения данных

Строковые инструкции перемещения данных во многом аналогичны инструкции MOV, но могут выполнять больше функций, чем инструкция MOV и работают быстрее. Мы рассмотрим сначала инструкцию LODS.

```
...
cld
mov si,0
lodsb
...
```

инструкция LODSB загружает регистр AL содержимым байта со смещением 0 в сегменте данных и увеличивает значение регистра SI на 1.

Это эквивалентно выполнению следующих инструкций:

```
...
mov si,0
mov al,[si]
inc si
...
```

Повторение строковой инструкции

Префикс повторения REP - это не инструкция, а префикс инструкции. Префикс инструкции изменяет работу последующей инструкции. Префикс REP делает следующее: он указывает, что последующую инструкцию нужно повторно выполнять до тех пор, пока содержимое регистра CX не станет равным 0. (Если регистр CX равен 0 в начале выполнения инструкции, то инструкция выполняется 0 раз, другими словами, никаких действий не производится.)

Задание. Прочитайте задание, которое выполняет нижеприведенная программа, напишите эту программу в эмуляторе. Напишите в отчете комментарии к программе и содержание регистров.

```
;BX сумма повторяющихся элементов обоих массивов
;dh количество одинаковых элементов
.model tiny
.stack 100h
.data
```

```
ArrayA db 05,10,06,44,20,32,05,11,46,0
ArrayB db 35,10,15,44,20,02,65,10,46,0
NuMofDiff dw 0 ;
NuMofEqual dw 0;
 sumO dw 0;
  sumR dw 0;
.code
start:
mov si,offset ArrayA
mov di,offset ArrayB
mov cx,10
 C1:
  push cx
  mov cx,10
  mov al,[si]
      C2:
      mov ah,[di]
      cmp al,ah
      jnz end_C2
      mov cl,ah
      add cl,ah
      add BX, cx
      inc DH
      jmp EX_C2:
      end_C2:
      inc di
      loop C2
  EX_C2:
  pop cx
  inc si
  mov di,offset ArrayB
 loop C1
end start
```

Задание 2. Напишите свою программу, которая сравнивала бы поэлементно буквы вашего имени и фамилии. В случае разницы размеров дополнить начальными буквами. Имя и фамилию написать заглавными латинскими буквами. Например, PAVLOVA ANNAANN (имя ANNA дополнено тремя буквами из самого себя ANN).

Подсчитать количество одинаковых и различных элементов, начиная с последнего в обратном порядке.

Контрольные вопросы:

1. Можно ли заменить команду STOSB командой STOSW?
2. Какую операцию производят строковые функции автоматически?
3. При применении строковых операций, можно ли в качестве регистра смещения использовать другие регистры, отличные от SI и DI?
4. Что служит результатом сравнения двух строк при помощи команды CMPS?

5. Используя, какую строковую инструкцию можно присвоить значение элементам массива?

Лабораторная работа №14

Тема: Процедуры

Цель работы: *ознакомиться с основами работы с процедурами, показать использование локальных переменных.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

В виде процедур обычно оформляются многократно повторяющиеся фрагменты программ для ускорения разработки и сокращения объема программ. Процедуры могут быть внутренними (находятся в том же модуле, что и вызывающая программа, транслируются совместно) и внешними (находятся в отдельных исходных модулях, транслируются отдельно). Последние чаще применяют в программах, предназначенных для преобразования в EXE-файлы, а также в случаях, когда процедуры достаточно велики по объему, сложны для отладки или могут быть использованы в других программах.

Ассемблер позволяет вам описывать процедуры несколькими способами. В данной работе описываются процедуры NEAR и FAR, объявление языка процедур, использование в процедурах аргументов и переменных, сохранение регистров, вложенные процедуры и описание процедур методов для объектов.

Синтаксис определения процедур

Для описания процедур вы можете использовать директиву PROC. Директива имеет следующий синтаксис:

```
PROC имя [расстояние]
[ARG список_аргументов] [RETURN список_элементов];
[LOCAL список_аргументов]
[USES список_элементов]
.
.
.
ENDP [имя]
```

Ассемблер также воспринимает для определения процедур синтаксис MASM.

Описание процедур NEAR или FAR

Процедуры NEAR вызываются с помощью вызова ближнего типа и содержат ближний возврат управления. Вы должны вызвать их только в том же сегменте, в котором они определены. Вызов ближнего типа заносит адрес возврата в стек и устанавливает указатель инструктор (IP) в значение смещения процедуры. Поскольку сегмент кода (CS) не изменяется, процедура должна находиться в том же сегменте, что и вызывающая

программа. Когда процессор обнаруживает возврат ближнего типа, он извлекает из стека адрес возврата и снова устанавливает в него IP. Сегмент кода не изменяется.

Процедура FAR вызывается с помощью вызова дальнего типа и содержит возврат дальнего типа. Процедура FAR вы можете вызывать вне сегмента. В котором они определяются. Вызов FAR заносит в стек адрес в виде сегмента и смещения, а затем устанавливается CS:IP в адрес процедуры. Когда процессор обнаруживает возврат дальнего типа, он извлекает из стека сегмент и смещение адреса возврата и устанавливает в него CS:IP.

В процедурах NEAR и FAR используется одна и та же инструкция RET. Ассемблер использует расстояние процедуры для определения того, требуется возврат ближнего и дальнего типа. Аналогично, ассемблер использует расстояние процедуры для определения того, требуется для ссылки на процедуру возврат ближнего или дальнего типа.

Работа команд RET и RETF

Алгоритм работы:

Работа команд зависит от типа процедуры;

- для процедур ближнего типа – восстановить из стека содержимое ip;
- для процедур дальнего типа – последовательно восстановить из стека содержимое ip и сегментного регистра cs;
- если команда ret имеет операнд, то увеличивать содержимое sp на величину операнда число.

Состояние флагов после выполнения команды не меняется.

Примечание: команду ret необходимо применять для возврата управления вызывающей программе из процедуры, управление которой было передано по команде call. На самом деле ЦП имеет три варианта команды возврата ret – это ret, ее синоним retn, а также команда retf. Они отличаются типами процедур, в которых используются. Команды ret и retn служат для возврата из процедур ближнего типа. Команда retf – команда возврата для процедур дальнего типа. Какая конкретно команда будет использоваться ,определяется компилятором; программисту лучше использовать команду ret, и доверить транслятору самому сгенерировать ее ближний или дальний вариант. Количество команд ret в процедуре должно соответствовать количеству точек выхода из нее. Некоторые языки высокого уровня, к примеру, Pascal, требуют, чтобы вызываемая процедура очищала стек от переданных ей параметром. Для этого команда ret содержит необязательный параметр число, который, в зависимости от установленного атрибута размера адреса, означает количество байт или слов, удаляемых из стека по окончании работы процедуры.

Синтаксис директив ARG и LOCAL

Приведем синтаксис определения передаваемых процедуре аргументов:

ARG аргумент [,аргумент]... [=идентификатор]
[RETURNS аргумент] [,аргумент]]

При определении локальных переменных процедуры используется следующий синтаксис:

LOCAL аргумент [,аргумент]... [=идентификатор]

Отдельные аргументы имеют следующий синтаксис:

имя_аргумента [[выражение_счетчик_1]]

[: сложный_тип [:выражение_счетчик_2]]

где "сложный_тип" - это тип данных аргумента. Он может быть либо простым типом, либо сложным выражением-указателем.

"Выражение_счетчик_2" задает, сколько элементов данного типа определяет аргумент. Например, в определении аргумента:

ARG tmp:DWORD:4

определяется аргумент с именем "tmp", состоящий из 4 двойных слов.

По умолчанию "выражение_счетчик_2" имеет значение 1 (кроме аргументов типа BYTE). Так как вы не можете занести в стек байтовое значение, для аргументов типа BYTE значение счетчика по умолчанию равно 2, что обеспечивает для них в стеке размер в слово.

Аргументы и переменные определяются в процедуре как операнды в памяти относительно BP. Передаваемые аргументы, определенные с помощью директивы ARG, имеют положительное смещение относительно BP. Локальные переменные, определенные с помощью директивы LOCAL, имеют отрицательное смещение от BP. Приведем пример:

```
...
func1 PROC NEAR
ARG a:WORD,b:WORD:4,c:BYTE=d
LOCAL x:DWORD,y=WORD:2=z
...
```

Здесь a определяется, как [bp+4], b определяется, как [bp+6], c определяется, как [bp+14], a d - как 20. x - это [bp-2], y - [bp-6], a z - 8.

Задание. Набрать нижеприведенную программу, которая переводит содержимое регистра AX в шестнадцатеричное представление и записывает результат в строку, смещение которой должно храниться в регистре BX.

```
.model tiny     ; Указывается тип памяти.
.stack 100h     ;Указывается размер памяти
.data   ;Начало сегмента данных
outStr db '0000$'  ;Выходная строка
SumOfDiff dw 0
SumOfEqual dw 0
```

```
NumOfDiff dw 0
NumOfEqual dw 0
.code  ;Начало программы
;Входные данные для процедуры translByte
;AL –байт, который нужно перевести
;Выходные данные
BX –смещение строки в первые два байта которой будет записан результат
translByte proc  ;Название процедурыpush ax
;Содержимое регистра AX помещаем в стек
push ax  ;Содержимое регистра AX помещаем в стек
shr al,4  ;Сдвигаем вправо на 4
cmp al,9  ;Сравнение
ja greater10 ;Если больше то поместить метку в греатер 10
mov byte ptr [bx],'0' ;Сделать значение BX 0.
add [bx],al  ;Добавить в регистра BX
jmp next4Bit  ;Элемент перехода.
greater10:  ;Метка греатер 10
mov byte ptr [bx],'A' ;Присвоить BX значение  A
sub al,10  ;Вычесть 10
add [bx],al  ;Добавить в BX
next4Bit:  ;Переход
pop ax  ;Выводим значение AX
and al,0Fh  ;Коньюнкция (И)
cmp al,9  ;Сравнение
ja _greater10 ;Если больше топ переместить в греатер 10
mov byte ptr [bx+1],'0';Функция вывода на экран
add [bx+1],al ;к BX+1 прибавить al
jmp exitByteProc ;Выход из процедуры
_greater10: ;Метка греатер 10
mov byte ptr [bx+1],'A';Сохранить
sub al,10  ;Вычесть 10
add [bx+1],al ;Сложить
exitByteProc: ;Выход из процедуры
pop ax  ;Выводит AX
ret  ;Возврат
translByte endp ;Завершение процедуры
```

```
translWord endp ;Завершение процедуры
start:  ;начало программы
mov ax,@data ;Настройка сегмента данных
mov ds,ax  ;Настройка сегмента данных
mov bx,OFFSET outStr;Функция ввода с клавиатуры
mov ax,60000 ;Ввод ax
call translWord  ;Вызов процедуры
mov ah,9  ;Вывод строки на экран
mov dx,OFFSET outStr;Ввод с клавиатуры
int 21h  ;Вывод прерывания для вызова строки
mov ax,4c00h ;Функция завершения программы
int 21h  ;Завершаем программу
end start  ;Конец программы с точки старт
```

Задание 2. Написать программу с использованием процедур, которая запрашивает строку (ввод с клавиатуры), и затем переводит все символы по следующему алгоритму:

Вариант 1. Если символ в нижнем регистре, перевести его в верхний регистр; если в верхнем – в нижний.

Вариант 2. Вывести строку в обратном порядке.

Вариант 3. Вывести строку, в закодированном виде, от каждого кода символа строки отнимается число 10.

Вариант 4. Удалить все символы в верхнем регистре.

Вариант 5. Найти позицию символа (вводится с клавиатуры) в строке, и вывести позицию (и) в шестнадцатеричном виде.

Контрольные вопросы:

1. Расскажите основные отличия между процедурами на языке Ассемблера от языков высокого уровня?
2. Расскажите основные отличия между адресацией процедур типа FAR и NEAR?
3. Какое смещение имеют локальные переменные, объявленные директивой LOCALS и почему?
4. Где применяется косвенный вызов процедур?
5. Что такое опережающий вызов процедуры?
6. Чем отличается косвенный вызов процедуры от прямого вызова?

Лабораторная работа №15

Тема: Сервисы DOS для работы с файловой системой

Цель работы: *закрепить навыки программирования циклов и работы с массивами на языке Ассемблер.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы

4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Пример 1. программа циклического повтора (LOOP)

```
#make_COM#

ORG 100h

mov cx, 10 ;заполнение счетчика

next:inc bx ;уменьшение счетчика

loop next ;проверка счетчика не 0

 ret
```

Пример 2. программа циклического повтора с проверкой условия (LOOPNZ)

```
#make_COM#

ORG 100h

mov bx, 3
mov cx, 10 ;заполнение счетчика

next:inc bx ;уменьшение счетчика

loopnz next ;проверка счетчика не равно 0

ret
```

Прерывания с номерами до INT 20H обслуживает BIOS. Номера прерываний DOS лежат в пределах от 20 до 32H:

Прерывания BIOS (без подфункций)

00h: Деление на ноль.
01h: Пошаговое.
02h: Немаскируемое.
03h: Точка прерыв.
04h: Переполнение.
ДОКУМЕНТАЦИИ)
05h: Печать экрана.
ДОКУМЕНТАЦИИ)
06h: (резерв)
печати)
07h: (резерв)
прерывания
08h: Таймер.
09h: Клавиатура.
0Ah-0dh: (hdwr ints)
0Eh: Дискета.
0Fh: (hdwr int)
10h: Видео сервис.
11h: Список оборудования.
12h: Размер исп.памяти.
13h: Дисковый в/в.
14h: В/в последовательный порт
15h: Расшир.сервис AT.
16h: В/в клавиатуры.

Прерывания DOS (без подфункций)

20h: Завершить программу
21h: Сервис DOS
25/26h: Абсолютные чтение/запись диска
27h: Завершиться, но остаться резидентным
28h: Квант времени DOS (НЕТ В

2eh: Выполнить команду DOS (НЕТ В

2fh: Мультиплексное прерывание (спулинг

Адресные указатели (не используются как

22h: Адрес завершения
23h: Адрес Control-Break
24h: Адрес обработчика критических ошибок

17h: В/в принтера.
18h: ROM-BASIC.
19h: Загрузка.
1Ah: В/в таймера.
1Bh: Прерывание клавиатуры.
1Ch: Прерывание по таймеру
1Dh: Видео параметры
1Eh: Параметры дискет
1Fh: Символы графики

Задание. Проверить работоспособность программы, производящей запись и чтение определенных значений из массива значений. Записать комментарии.

```
#make_COM#
ORG  100H

.data

mes db 0ah,0dh,'Массив-','$'
mas db 1,2,3,4,5,6
i db 0
.code

main:
xor ax,ax
mov cx, 10
mov si, 0

go:
mov bh,i
mov mas[si],bh
inc i
inc si
loop go
mov cx,10
mov si,0
mov ah,09h
lea dx,mes
int 21h

show:
mov ah,02h
mov dl,mas[si]
add dl,30h
int 21h
inc si
loop show

exit:   MOV AH,4Ch
        INT 21h
end main

```

Задание 2. Проверить работоспособность программы, производящей запись и чтение значений из пустого массива элементов.

```
ORG  100H

.data
;n=5
;mes db 0ah,0dh,'Массив-','$'
mas db 5 dup (3 dup (0))
i db 0
.code

main:
mov ax,@data
mov ds,ax
xor ax,ax
mov cx,5
mov si,0

go:
mov dl,mas[si]
inc dl
mov mas[si],dl
add si,3
loop go
mov cx,5
mov si,0

show:
mov dl,mas[si]
add dl,30h
mov ah,02h
int 21h
inc si
loop show

exit:   MOV AH,4Ch
        INT 21h
end main
```

Задание 3. Дан текст, состоящий из слов, разделенных пробелами. Определить количество повторений в тексте заданной буквы. Написать комментарии.

```
org 100h
start: mov ah, 09h
       lea dx, STRING
       int 21h
mm:             CLD
                MOV   AL,'1'
                LEA   DI, STRING
                MOV   CX,27
                REPNE SCASB
                DEC   DI
jcxz exit
mov bx,26
sub bx,cx
mov al,STRING[bx]

add kol,1
xor al,al
mov STRING[bx],al
jmp mm
```

```
exit:
mov dl,al
mov ah,02h
int 21h
add kol,48
MOV  AL,kol
mov dl,al
mov ah,02h
int 21h

mov ah, 4ch
int 21h
  STRING db 'hello, aauorldASTRING perl', 0ah, 0dh, '$'
  kol db 0
end start

ret
```

Задание 4. Разработать программу, вычисляющую номер нулевого элемента массива.

```
org 0x100
.data
mes db 0ah,0dh,'Array- ','$'
mas db 1,2,3,4,5,6
i db 0
.code
mov cx, 6
mov si,0
mov ah,09h
lea dx,mes
int 21h
mov ah,02h
show:
mov dl,mas[si]
add dl,30h
int 21h
inc si
loop show


mov ah,4ch
int 21h
```

Контрольные вопросы:

5. Что такое массив?
6. Назначение команды loopnz?
7. Назначение функции 4ch?
8. Назначение функции 09h?

Лабораторная работа №16

Тема: Разработка подпрограмм и программных прерываний средствами языка Ассемблер

Цель работы: *сформировать навыки программирования подпрограмм на языке Ассемблер.*

Ход работы:

1. Выполнить задание в соответствии с указаниями
2. Ответить на контрольные вопросы
3. Предъявить преподавателю результаты работы: проект и исходный код
4. Оформить отчет в соответствии с ходом работы

Схематичный алгоритм программы:

Таблица 1.

Шаг	Алгоритм главной программы
1	Запись Y в регистр DX
2	Запись X в регистр BX
3	Запись R в регистр AX
4	Запись в стек содержимого регистра BX
5	Запись в стек содержимого регистра DX
6	Запись в стек содержимого регистра AX
7	Вызов подпрограммы PODPR
8	Восстановление из стека регистра BX
9	Восстановление из стека регистра DX
10	Восстановление из стека регистра AX
Шаг	Алгоритм подпрограммы PODPR
1	Запись в стек содержимого регистра BP
2	Сохранение значения BP (BP:= SP)
3	Считывание из стека X
4	Возведение X в квадрат
5	Считывание из стека Y
6	Возведение Y в квадрат
7	Запись в регистр CX значение X^2+Y^2
8	Считывание из стека R
9	Возведение R в квадрат
10	Запись в регистр AX значение R^2
11	Сравнение регистров CX и AX
12	Если CX>AX то запись в регистр AX значение 1 иначе 0
13	Восстановление из стека регистра BP

Задание. Проверить работоспособность программы производящей вычисление $Y = N!–M!$, где N и M факториал числа 5 и 4.

```
#make_com#

org  100h

.data
n    dw 5
m    dw 4
y    dw ?

.code

begin:
mov ax,@data
mov ds,ax

mov cx,m     ;cx:=m
call Factor ;вызов подпрограммцы для вычисления m!
mov bx,ax    ;bx:=m!
mov cx,n     ;cx:=n
call Factor ;вызов подпрограммцы для вычисления n!
sub ax,bx    ;ax:=n!-m!
mov y,ax     ;y:=ax
mov ah,4ch  ;выход из программы
int 21h

Factor proc      ;начало подпрограммы
mov ax,1     ;ax:=1
for:
mul cx       ;ax:=ax*cx
loop for         ;cx:=cx-1
ret ;возврат в основную программу

Factor endp      ;конец подпрограммы

end begin

ret

```

Задание 2. Разработать программу, вычисляющую $Y = M! + №!$, где №! - порядковый номер по журналу факториала, М! = 4.

Контрольные вопросы:

1. Что такое подпрограмма?
2. С помощью, какой команды можно сравнить значения в регистрах?
3. С помощью какой команды можно вызвать подпрограмму?
4. С помощью какой команды можно вернуться в основную программу?

Лабораторная работа №17

Тема: Разработка подпрограмм и программных прерываний средствами языка Ассемблер

Цель работы: *сформировать навыки работы с подпрограммами при программировании на языке Ассемблер.*

Ход работы:

1. Выполнить задание в соответствии с указаниями
2. Ответить на контрольные вопросы
3. Предъявить преподавателю результаты работы: проект и исходный код

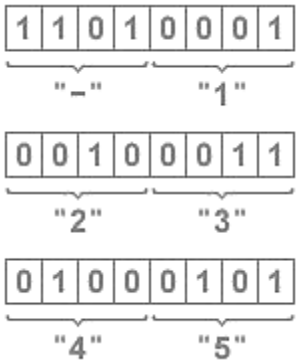
4. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

В настоящее время распространены два формата представления десятичных чисел в микропроцессорах - упакованный двоично-десятичный код (BCD-Binary-Coded Decimal) и неупакованный десятичный код .

Упакованный BCD-код - это такое представление десятичного числа, когда каждая десятичная цифра представляется 4-х битным двоичным позиционным кодом 8-4-2-1. При этом байт содержит две десятичные цифры. Младшая десятичная цифра занимает правую тетраду (биты 3 : 0), старшая - левую тетраду (биты 7 : 4). Многоразрядные BCD-числа занимают несколько смежных байт.

Если число является знаковым, то для представления знака в BCD-формате отводится старшая тетрада старшего байта. Для кодирования знака можно использовать шесть двоичных кодовых комбинаций, которые не используются для представления десятичных цифр. Это коды 1010-1111 (A-F в шестнадцатеричном представлении). Обычно для кодирования знака плюс применяют код 1100 (C), а для знака минус - 1101 (D).



Неупакованный десятичный код является подмножеством международной таблицы кодирования символов ASCII (Таблица 1). Видно, что для хранения неупакованных десятичных чисел требуется в два раза больше памяти, так как каждая цифра представляется 8-битным кодом. Таблица 1: ASCII-коды десятичных цифр

Неупакованный десятичный код является подмножеством международной таблицы кодирования символов ASCII (Таблица 1). Видно, что для хранения неупакованных десятичных чисел требуется в два раза больше памяти, так как каждая цифра представляется 8-битным кодом.

Десятичная цифра	ASCII-код	Десятичная цифра	ASCII-код
0	$30	5	$35
1	$31	6	$36
2	$32	7	$37
3	$33	8	$38
4	$34	9	$39

Таблица 1: ASCII-коды десятичных цифр

В языке ассемблера не существует специальных средств для организации работы с массивами. Массивы описываются в виде последовательности элементов нужной размерности; работа с ними ведется с использованием методов косвенной адресации.

Пример: Задан массив. Вывести на экран сумму его элементов.

```
1  data segment
2        mas db 12, 2, 7, 3, 2, 0,
         21, 9, 0, 16          ;исходный массив
3        len dw $-mas          ;размер массива в байтах
4  data ends
5
6  code segment
7  start:
8        assume cs:code, ds:
         data
9        mov ax, data
10       mov ds, ax
11
12       lea bx, mas           ;берём в bx адрес первого элемента
13       mov cx, len           ;счетчик цикла
14       xor ax, ax
15 cikl: add al,[bx]           ;прибавляем к al байт, адрес которого
                               хранится вbx
16       inc bx                ;переходим к следующему элементу
17       loop cikl
18
19       aam                   ;преобразуем сумму в BCD-код
20       add ax, 3030h         ;затем в ASCII-код
21       mov bx, ax            ;и выводим
22       mov ah, 02
23       mov dl, bh
24       int 21h
25       mov dl, bl
26       int 21h
27
28       mov ax, 4c00h
29       int 21h
30 code ends
31 end start
```

При обработке не байтовых массивов следует учитывать размер элементов. Так в приведенном выше примере в строке 3 вычисляется размер массива в байтах, и в строке 13 он берётся за количество элементов. Для массива слов это значение придется поделить на 2. В строке 16 для перехода к следующему элементу используется простой инкремент адреса, для массива слов эта строка может выглядеть так:`add bx, 2`.

Задание. Подсчитать в массиве количество элементов, равных введённому N.

Задание 2. Заменить в массиве все четные элементы нулями.

Задание 3. Определить номер первого равного нулю элемента.

Контрольные вопросы:

1. Для каких данных целесообразно использовать массивы?
2. Как организуется работа с массивами в языках высокого уровня?
3. Какие алгоритмы существуют для сортировки массивов и поиска элементов в них?

Лабораторная работа №18

Тема: Копирование файлов с использованием функции полуфабриката Win32

Цель работы: *сформировать навыки работы с подпрограммами при программировании на языке Ассемблер.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Win32 содержит множество функций-полуфабрикатов, которые объединяют несколько функций для выполнения часто встречающихся задач. В некоторых случаях эти функции-полуфабрикаты могут улучшить быстродействие. Например, CopyFile значительно упрощает программу. В частности, здесь совершенно не нужно искать оптимальный размер буфера, который в двух предыдущих программах был произвольно задан в 256 байт.

Реализация программы копирования файлов для Win32 с использованием CopyFile достаточно проста и обеспечивает высокое быстродействие.

Обращение производится из командной строки: cpCF файл1 файл2 - копирует файл1 в файл2.

```
#include
#include
int main (int argc, LPTSTR argv [])
{
if (argc != 3)
{
printf (“Использование: cpCF file1 file2\n”);
return 1;
}
if (!CopyFile (argv [1], argv [2], FALSE))
{
```

```
printf (“Ошибка CopyFile: %x\n”, GetLastError ());

return 2;
}
return 0;
}
```

Приведенные программы копирования файлов демонстрируют многочисленные различия между библиотекой C и Win32. Примеры с использованием Win32 демонстрируют стиль программирования и соглашения Win32, но дают лишь первое представление о функциональных возможностях, доступных в Win32.

Задания

1. Изучить программы копирования файлов.
2. Последовательно набрать и отладить программы копирования файлов в среде Visual Studio C++ .NET 2003.
3. Выполнить задание по копированию файла 1 в файл 2, выбрав произвольные имена файлов в различных программах.
4. Подготовить отчет по выполненной работе.
5. В отчете ответить на контрольные вопросы.

Порядок выполнения программ в среде Visual C++

1. Запустить Microsoft Visual Studio C++ .NET 2003.

2. Создать проект под именем cpC. Для этого выполнить команды меню File, New, Project.

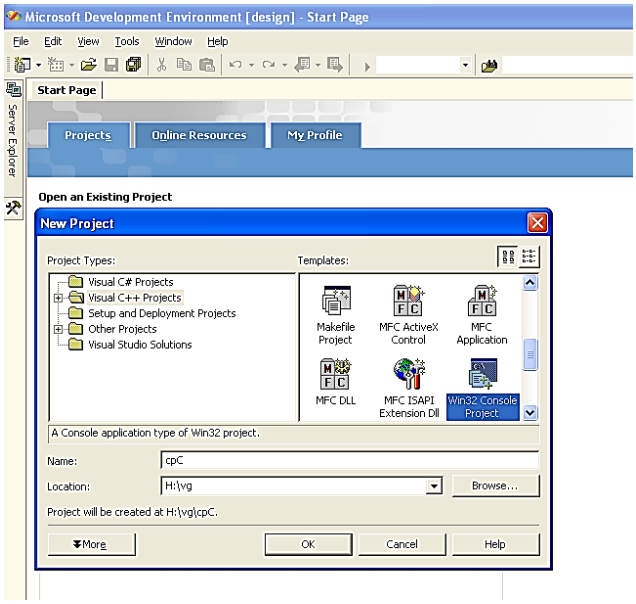


Рис.1. Создание нового проекта cpC консольного приложения

На появившейся вкладке ^ New Project выбрать Win32 Console Project, в окне Name: набрать имя проекта cpC. В окне Location: набрать путь к папке, в которой будет записан проект и щелкнуть мышью OK. На появившейся вкладке щелкнуть мышью Finish.

На экране появляется текст файла cpC.cpp, в котором находится заготовка программы консольного приложения.

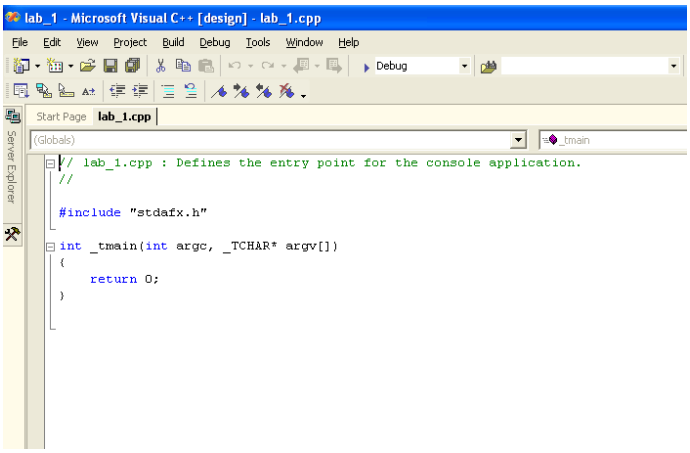


Рис 2. Содержимое файла cpC.cpp

После набора текста программы выполняется ее редактирование с использованием из меню Build команды Build Solution. В случае исправления всех ошибок в папке Debug проекта создается исполняемый файл с расширением .exe - cpC.exe, который необходимо выполнить.

Выполнение программы производится из командной строки.

Для этого можно использовать программу Far, при этом следует указать имя исходного файла, который должен быть создан до запуска программы cpC.exe, и имя файла, куда производится копирование.

Аналогично выполняется создание проектов cpW, cpCF, создание исполняемых файлов cpW.exe, cpCF.exe и выполнение программ копирования файлов с использованием Win32 и фунции-полуфабриката Win32.

Контрольные вопросы

1. Что такое API Win32?

2. Какие операционные системы обслуживает API Win 32?

3.Какие особенности имеет API Win 32?

4. Какие преимущества программирования дает API 32?

5. Какой основной тип переменных используется в Win 32?

6. Для управления каких систем могут быть написаны программы с использованием Win32?

7. Что означает строка int main (int argc, LPTSTR argv [])?

Лабораторная работа №19

Тема: Вывод списка файлов и их атрибутов в заданном каталоге

Цель работы: *сформировать навыки работы с файлами.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

1. Управление каталогами

Создание или удаление каталога выполняется двумя простыми функциями.

BOOL CreateDirectory (LPCTSTR lpszPath,LPSECURITY_ATTRIBUTES lpsa)

BOOL RemoveDirectory (LPCTSTR lpszPath)

lpszPath - указывает на строку с завершающим нулем, содержащую имя каталога, который должен быть создан или удален. Второй параметр (атрибуты безопасности) пока должен быть равен NULL. Удалить можно только пустой каталог.

Каждый процесс имеет текущий, или рабочий, каталог. Программист может и узнать текущий каталог, и установить его. Функция SetCurrentDirectory устанавливает каталог:

BOOL SetCurrentDirectory (LPCTSTR lpszCurDir)

lpszCurDir - путь к новому текущему каталогу. Это может быть относительный путь или полностью уточненный путь, начинающийся либо с буквы диска и двоеточия, например D:\.

Если путь к каталогу - просто обозначение привода (например, A: или C:), текущим становится рабочий каталог указанного диска. Например, если рабочие каталоги установлены в последовательности

C:\MSDEV

INCLUDE

то в результате текущий каталог будет C: \MSDEV\INCLUDE.

Функция GetCurrentDirectory возвращает в буфер, заданный программистом, полностью уточненный путь.

DWORD GetCurrentDirectory (DWORD cchCurDir,LPTSTR lpszCurDir)

Возвращаемое значение - длина строки возвращенного пути или требуемый размер буфера, если он недостаточно велик; нуль при неудаче.

cchCurDir - длина в символах (не в байтах) буфера для имени каталога. Длина должна учитывать завершающий нулевой символ,

lpszCurDir - указывает на буфер, в который записывается строка пути.

Если буфер слишком мал для пути, возвращаемое значение сообщает его необходимую величину. Поэтому, при проверке на неудачное выполнение функции должен проверяться как нулевой результат функции, так и результат, превышающий значение параметра cchCurDir.

Атрибуты файлов и работа с каталогами

В каталоге можно искать файлы и другие каталоги, удовлетворяющие указанному образцу имени, а также получать атрибуты файлов. Для этого необходим дескриптор поиска, который возвращается функцией FindFirstFile. Для получения нужных файлов служит функция FindNextFile, а для завершения поиска - FindClose.

HANDLE FindFirstFile (LPCTSTR lpszSearchFile, LPWIN32_FIND_DATA lpffd);

Возвращаемое значение: дескриптор поиска .

INVALID_HANDLE_VALUE указывает на неудачу.

FindFirstFile ищет соответствие имени как среди файлов, так и среди подкаталогов. Возвращаемый дескриптор используется в последующем поиске.

lpszSearchFile указывает на каталог или полное имя, которое может содержать подстановочные знаки (? и *).

lpffd указывает на структуру WIN32_FIND_DATA, которая содержит информацию о найденном файле или каталоге.

Структура WIN32_FIND_DATA определена следующим образом:

```
typedef  struct WIN32_FIND_DATA  {
DWORD dwFileAttributes;
FILETIME   ftCreationTime;
FILETIME  ftLastAccessTime;
FILETIME  ftLastWriteTime;
DWORD nFileSizeHigh;
DWORD nFileSizeLow;
DWORD dwReserved0 ;
DWORD dwReservedl;
TCHAR cFileName [MAX_PATH];
TCHAR cAlternateFilename   [14 ];}
WIN32 _FIND_DATA;
```

dwFileAttributes можно проверить на соответствие значениям, приведенным в описании CreateFile. Далее следуют три значения времени (время создания файла, время последнего доступа и последней записи). Поля размера файла, содержащие его текущую длину, не требуют пояснений,

cFileName - это не полное имя, а собственно имя файла,

cAlternateFile - версия имени файла для DOS формата 8.3 (включая точку). Это то же сокращение имени файла, которое отображается в проводнике Windows. Оба имени - строки с завершающим нулем.

Часто требуется найти в каталоге файлы, имена которых удовлетворяют образцу, содержащему подстановочные знаки ? и *. Для этого нужно получить из FindFirstFile дескриптор поиска, который содержит информацию об искомом имени, и вызвать функцию FindNextFile.

BOOL FindNextFile (HANDLE hFindFile, LPWIN32_FIND_DATA lpffd);

Функция FindNextFile возвращает FALSE либо при недопустимых параметрах, либо если соответствие данному образцу уже нельзя найти. В последнем случае GetLastError возвращает ERROR_NO_MORE_FILES.

Когда поиск закончен, необходимо закрыть дескриптор поиска. Нельзя использовать для этого функцию CloseHandle. (Это редкое исключение из правила, по которому CloseHandle пригодна для закрытия всех дескрипторов; закрытие дескриптора поиска вызовет исключительную ситуацию) Для закрытия дескрипторов поиска предназначена функция FindClose:

BOOL FindClose (HANDLE hFindFile);

Функция GetFileInformationByHandle позволяет получить ту же информацию для определенного файла, указав его открытый дескриптор.

Функции FindFirstFile и FindNextFile позволяют получать следующую информацию об атрибутах файла: флаги атрибутов, три штампа времени и размер файла. Для работы с атрибутами есть несколько других функций, включая ту, что позволяет их устанавливать, и они могут работать непосредственно с дескриптором открытого файла без просмотра каталога или указания имени файла. Другие функции служат для получения остальных атрибутов.

Функция GetFileAttributes по указанному имени файла и каталога возвращает только атрибуты.

DWORD GetFileAttributes (LPCTSTR lpszFileName);

Возвращаемое значение: атрибуты файла или OxFFFFFFFF в случае неудачи.

Атрибуты можно проверить на соответствие комбинациям определенных значений. Некоторые атрибуты, такие как флаг временного файла, устанавливаются при вызове CreateFile. Значения атрибутов могут быть следующими:

FILE_ATTRIBUTE_DIRECTORY

FILE_ATTRIBUTE_NORMAL

FILE_ATTRIBUTE_READONLY

FILE_ATTRIBUTE_TEMPORARY

Функция SetFileAttributes изменяет эти атрибуты в указанном по имени файле.

Теперь поясним функции управления файлами и каталогами. В листинге приведена версия программы вывода содержимого каталога, которая показывает время изменения файла и его размер, хотя может отображать только младшую часть значения размера файла.

Программа ищет в каталоге файлы, которые удовлетворяют образцу поиска. Для каждого найденного файла программа показывает имя и, если указана опция -1, атрибуты файла.

Большая часть листинга посвящена обходу каталога. Каждый каталог программа проходит дважды: один раз для обработки файлов и второй - для обработки подкаталогов, обеспечивая рекурсивную работу, когда указана опция – R.

Программа правильно работает при указании абсолютного пути, например:

<Имя программы> -R -1 C:\test*.txt. Программа обеспечит вывод списка файлов с расширением .txt и их атрибутов из каталога C:\test и подкаталогов.

Программа вывод списка файлов и обход каталога

```
//Директивы препроцессора
#include "stdafx.h"
//#include "EvryThng.h"
//Далее приводится текст библиотеки «EvryThng.h»
#include "stdafx.h"
#define WIN32_LEAN_AND_MEAN
#define NOATOM
#define NOCLIPBOARD
#define NOCOMM
#define NOCTLMGR
#define NOCOLOR
#define NODEFERWINDOWPOS
#define NODESKTOP
```

```
#ifdef UNICODE
#define _UNICODE
#endif
#ifndef UNICODE
#undef _UNICODE
#endif
//#define _MT
#define _STATICLIB
/* Определяйте _STATICLIB при формировании статической
библиотеки или компоновке с ней. */
//#undef_STATICLIB
#include <windows.h>
#include <tchar.h>

#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <search.h>
#include <string.h>
/*  "Support.h"  */
/* Определения всех символьных констант и служебных функций
общего применения для всех программ примеров. */
/* ЛУЧШЕ ОПРЕДЕЛИТЬ СИМВОЛЫ UTILITY_EXPORTS И
_STATICLIB НЕ ЗДЕСЬ, А В ПРОЕКТЕ, ОДНАКО ИХ ОПИСАНИЯ
ПРИВОДЯТСЯ ЗДЕСЬ. */
/* Символ "UTILITY_EXPORTS" генерируется в Dev Studio при
создании проекта DLL с именем "Utility" и определяется в командной
строке C. */
//#define UTILITY_EXPORTS Закомментировано; определяйте
//       этот символ в проекте.
#define UTILITY_EXPORTS
#ifdef UTILITY_EXPORTS
#define LIBSPEC _declspec (dllexport)
#else
#define LIBSPEC _declspec (dllimport)
#endif
#define EMPTY _T ("")
#define YES _T ("y")
#define NO _T ("n")
#define CR OxOD
#define LF OxOA
#define TSIZE sizeof (TCHAR)
```

```
/* Пределы и константы. */
#define TYPE_FILE 1   /* Используется в ls, rm и lsFP. */
#define TYPE_DIR 2
#define TYPE_DOT 3
#define MAX_OPTIONS 20
/* Максимальное кол-во опций командной строки. */
#define MAX_ARG 1000
/* Максимальное кол-во параметров командной строки. */
#define MAX_COMMAND_LINE MAX_PATH+50
/* Максимальный размер командной строки. */
/* Функции общего применения. */
LIBSPEC BOOL ConsolePrompt (LPCTSTR, LPTSTR, DWORD, BOOL);
LIBSPEC BOOL PrintStrings (HANDLE, ...);
LIBSPEC BOOL PrintMsg (HANDLE, LPCTSTR);
LIBSPEC VOID ReportError (LPCTSTR, DWORD, BOOL);
```

//Начало программы

```
BOOL TraverseDirectory (LPCTSTR, DWORD, LPBOOL);
DWORD FileType (LPWIN32_FIND_DATA);
BOOL ProcessItem (LPWIN32_FIND_DATA, DWORD, LPBOOL);
```

DWORD Options (int argc, LPCTSTR argv [], LPCTSTR OptStr, ...)

/* argv - командная строка. Опции, начинающиеся с '-', если они есть, помещаются в argv [1], argv [2], ...

OptStr - текстовая строка, содержащая все возможные опции во взаимно однозначном соответствии с адресами булевых переменных в списке параметров-переменных (...).

Эти флаги устанавливаются тогда и только тогда, когда символ соответствующей опции встречается в argv [I], argv [2], ...

Возвращаемое значение - индекс в argv первого параметра, не принадлежащего к опциям. */

Задание.

1. Запустить Microsoft Visua Studio C++ .NET 2003.

2. Создать проект под именем **lab2**. Для этого выполнить команды меню **File, New**, **Project.** На появившейся вкладке выбрать **Win32 Console Project**, в окне **Name:** набрать имя проекта **lab2** и щелкнуть мышью OK. На появившейся вкладке щелкнуть мышью **Finish**. Если файла lab2.cpp на экране нет, то в меню **View** выбрать **Solution Explorer**. В папке **1ab2 files** выбрать папку **Source Files** и открыть файл программы **lab2.cpp**.

3.После набора текста программы выполняется ее компиляция с использованием меню **Build** команды **Rebuild Solution**. В случае исправления всех ошибок в папке **Debug** проекта создается исполняемый файл с расширением **lab2.exe**, который необходимо выполнить.

4. Выполнение указанной программы производится из командной строки для 3-х вариантов: поиск файлов в папке Debug по заданному шаблону; поиск файлов по заданному шаблону с выводом атрибутов файла, задавая ключ -1; поиск файлов позаданному шаблону в директории, созданной в папке Debug.

Контрольные вопросы:

1. Поясните задачу функции: FindFirstFile и ее параметры.
2. Поясните задачу функци: FindNextFile и ее параметры.
3. Какова структура параметра lhffd?
4. Какой параметр функции FindFirstFile указывает на каталог?

5. Какие существуют функции для работы со временем?

Лабораторная работа №20

Тема: Копирование нескольких файлов в стандартный вывод

Цель работы: *сформировать навыки работы с файлами.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

1. Стандартные устройства Win32

Win32 имеет три стандартных устройства для ввода, вывода и сообщения об ошибках, для которых требуются дескрипторы.

Имеется функция для получения дескрипторов стандартных устройств.

HANDLE GetStdHandle (DWORD nStdHandle)

Возвращаемое значение: допустимый дескриптор, если функция завершается успешно; иначе INVALID _HANDLE _VALUE.

Параметры GetStdHandle

nStdHandle должен иметь одно из следующих значений:

• STD_INPUT_HANDLE;

• STD_OUTPUT_HANDLE;

• STD_ERROR_HANDLE.

Стандартные устройства обычно назначены консоли и клавиатуре. Стандартный ввод-вывод можно перенаправлять.

GetStdHandle не создает новый дескриптор стандартного устройства и не дублирует прежний. Последовательные вызовы с одним и тем же параметром возвращают одно и то же значение дескриптора. Закрытие дескриптора стандартного устройства делает это устройство недоступным для последующего использования, поэтому в программах часто открывают стандартное устройство, работают с ним, но не закрывают. Назначение стандартных устройств производится функцией:

BOOL SetStdHandle (DWORD nStdHandle, HANDLE hHandle)

Возвращаемое значение: TRUE или FALSE, в зависимости от успеха, или неудачи.

Параметры SetStdHandle

Возможные значения nStdHandle - те же, что в GetstdHandle. Параметр hHandle определяет открытый файл, который должен быть стандартным устройством.

Обычный метод перенаправления стандартного ввода-вывода в пределах процесса состоит в том, чтобы использовать последовательность SetStdHandle и GetStdHandle. Полученный в результате дескриптор используется в последующих операциях ввода-вывода.

Два имени файлов зарезервированы для консольного ввода (с клавиатуры) и консольного вывода: CONIN$ и CONOUT$. Первоначально стандартный ввод, вывод и выдача ошибок назначены на консоль. Консоль можно использовать независимо от

любого перенаправления этих стандартных устройств; для этого нужно просто открыть дескрипторы CONIN$ ИЛИ CONOUT$ С ПОМОЩЬЮ CreateFile.

Для консольного ввода-вывода можно применить ReadFile и WriteFile, но лучше использовать специальные функции ReadConsole и WriteConsole. Основные их преимущества состоят в том, что эти функции обрабатывают универсальные символы (TCHAR), а не байты, а также учитывают режим консоли, установленный функцией SetConsoleMode.

BOOL SetConsoleMode (HANDLE hConsole, DWORD f devMode)

Возвращаемое значение: TRUE, если функция завершается успешно.

hConsole - определяет буфер ввода или экран, который должен иметь аттрибут доступа GENERIC_WRITE,

fdevMode - определяет режим обработки символов. Значение каждого флага указывает, применяется ли этот флаг к консольному вводу или выводу.

Программа копирования нескольких файлов в стандартный вывод

/*При наличии в командной строке запуска программы имени одного или нескольких файлов программа перенаправляет их содержимое на экран. Если имя файла отсутствует, данные вводятся с клавиатуры и перенаправляются на экран */

```
#include "stdafx.h"
#include "EvryThng.h"
#define BUF_SIZE 0x200
static VOID CatFile (HANDLE, HANDLE);
DWORD Options (int argc, LPCTSTR argv [], LPCTSTR OptStr, ...)
```

/* argv - командная строка. Опции, начинающиеся с '-', если они есть, помещаются в argv [1], argv [2], ...

OptStr - текстовая строка, содержащая все возможные опции во взаимно-однозначном соответствии с адресами булевых переменных в списке параметров-переменных (...). Эти флаги устанавливаются тогда и только тогда, когда символ соответствующей опции встречается в argv [1], argv [2], ...

Возвращаемое значение - индекс в argv первого параметра, не принадлежащего к опциям. */

```
{
va_list pFlagList;
LPBOOL pFlag;
int iFlag = 0, iArg;
va_start (pFlagList, OptStr) ;
while ((pFlag = va_arg (pFlagList, LPBOOL)) != NULL && iFlag < (int)
_tcslen (OptStr))
{
*pFlag = FALSE;
for (iArg = 1; !(*pFlag) && iArg < argc && argv [iArg] [0] == '-';iArg++)
*pFlag = _memtchr (argv [iArg], OptStr [iFlag],
_tcslen (argv [iArg])) != NULL; iFlag++;
}
```

Задание.

1. Запустить Microsoft Visua Studio C++ .NET 2003.

2. Создать проект под именем **lab3**. Для этого выполнить команды меню **File, New**. На появившейся вкладке выбрать **Win32 Console Project**, в окне **Name:** набрать имя проекта **lab3** и щелкнуть мышью OK. На появившейся вкладке щелкнуть мышью по кнопке **Finish**. После щелчка по папке Project в правой части окна MDE отобразится ее содержимое: 3папки – **Source Files** (исходные файлы), **Header Files** (заголовочные файлы)

и **Resource Files** (файлы ресурсов), а также файл ReadMe.txt. Откройте папку **Source Files** и дважды щелкните на файл .cpp. Тогда откроется окно, в котором выполняется набор текста программы.

3.После набора текста программы выполняется ее редактирование с использованием меню **Build** команды **Rebuild Solution**. В случае исправления всех ошибок в папке **Debug** проекта создается исполняемый файл **lab3.exe**, который необходимо выполнить. Выполнение указанной программы производится из командной строки, задавая при этом после имени программы имя файла, который копируется на стандартное устройство вывода.

Контрольные вопросы:

1. Поясните задачу функции: FindFirstFile и ее параметры.
1.Поясните функцию: GetStdHandle и ее параметры.
2.Поясните функцию: SetStdHandle и ее параметры.
3.Поясните функцию: SetConsoleMode и ее параметры.
4.Какую задачу выполняет функция VOID CatFile?
5.Как составляется командная строка при перенаправлении вводимых с клавиатуры символов на экран?

Лабораторная работа №21

Тема: Последовательная обработка файлов с использованием отображения.
Цель работы: *сформировать навыки работы с файлами.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

1.Создание объекта отображения файла

Объекты отображения файлов в память могут получать имена, поэтому они доступны другим процессам для разделения памяти. Также отображаемый объект имеет защиту, атрибуты безопасности и размер.

Для создания объекта отображения в память используется функция:

HANDLE CreateFileMapping (HANDLE hFile,LPSECURITY_ATTRIBUTES lpsa, DWORD fdwProtect; DWORD dwMaximumSizeHigh, DWORD dwMaximumSizeLow, LPCTSTR lpszMapName);

Возвращаемоезначение:дескриптор отображения файла или NULL при неудаче.

hFile — дескриптор открытого файла с флагами защиты, совместимыми с параметром fdwProtect.

Тип LPSECURITY_ATTRIBUTES позволяет защитить объект отображения. Параметр fdwProtect определяет доступ к отображенному файлу с помощью описанных ниже флагов. Для специальных целей разрешены дополнительные флаги.

Установленный флаг PAGE_READONLY означает, что программа может только читать страницы в отображенной области и не может записывать или исполнять их. Файл hFile должен быть открыт с правом доступа GENERIC_READ.

• Флаг PAGE_READWRITE предоставляет полный доступ к объекту, если файл hFile открыт с правами доступа GENERIC_READ и GENERIC_WRITE.

• Флаг PAGE_WRITECOPY определяет, что при изменении содержимого отображенной памяти собственная (для данного процесса) копия записывается в файл подкачки, а не в исходный файл.

• Параметры dwMaximumSizeHigh и dwMaximumSizeLow определяют размер объекта отображения. Если указан нуль, используется текущий размер; при использовании файла подкачки унеобходимо обязательно определять его размер. Если ожидается, что размер файла увеличится, необходимо использовать ожидаемый размер, и при необходимости будет немедленно установлен нужный размер файла. Нельзя отображать область файла за указанной границей - объект отображения не может расти.

• lpszMapName указывает имя объекта отображения, что позволяет другим процессам совместно использовать объект. Регистр символов в имени не учитывается. Если разделение памяти не используется, указывается значение NULL.

Об ошибке сообщает возвращаемое значение NULL (а не INVALID_HANDLE_VALUE).

Указав имя существующего объекта отображения, можно получить дескриптор отображения файла. Имя должно быть получено предшествующим вызовом функции CreateFileMapping. Два процесса могут совместно использовать память, разделяя отображение файла. Первый процесс создает отображение файла, а следующий открывает это отображение, используя имя. Если названного объекта не существует, открыть его не удастся.

```
HANDLE OpenFileMapping (
    DWORD dwDesiredAccess,
    BOOL bInheritHandle,
    LPCTSTR lpName);
```

Возвращаемое значение: дескриптор отображения.....файла или NULL при неудаче.

dwDesiredAccess использует тот же набор флагов, что и параметр fdwProtect функции CreateFileMaping.

lpName - имя, полученное вызовом функции CreateFileMapping.

Наследование дескриптора - параметр bInheritHandle.

Функция CloseHandle уничтожает дескрипторы отображения.

2.Выделение виртуального адресного пространства для объекта отображения

Следующий этап - выделение виртуального адресного пространства и отображение его в файл через объект отображения. С точки зрения программиста, такое выделение памяти аналогично действию функции HeapAlloc, но намного грубее, более крупными частями. Возвращается указатель на выделенный блок (или образ файла); отличие состоит в том, что выделенный блок отображается в указанный пользователем файл, а не в файл подкачки обмена.

```
LPVOID MapViewOfFile(
    HANDLE hMapObject,
    DWORD  fdwAccess,
    DWORD dwOffsetHigh,
    DWORD dwOffsetLow,
    SIZE_T  cbMap);
```

Возвращаемое значение: начальный адрес блока (образ файла) или NULL при неудаче.

hMapObject - указывает объект отображения файла, полученный от функций CreateFileMapping или OpenFileMapping.

Значение параметра fdwAccsess должно соответствовать правам доступа объекта отображения. Возможны три значения флагов:

```
FILE_MAP_WRITE,
FILE_MAP_READ,
FILE_MAP_ALL_ACCESS.
```

Параметры dwOffsetHigh и dwOffsetLow определяют начальное положение области отображения. Начальный адрес должен быть кратным 64К. Для отображения от начала файла используется нулевое значение смещения.

cbMap указывает размер отображаемой области в байтах. Его нулевое значение при вызове функции MapViewOfFile указывает на то, что весь файл должен быть отображен.

Освобождение образа файла выполняется так же, как и освобождение памяти, выделенной в куче, т.е. функцией HeapFree.

3.Программа работы с файлом, отображенным в память

Основным преимуществом отображения в память является возможность использования удобных алгоритмов для обработки файлов.

В приведенной программе осуществляется отображение файла в память. Имя файла задается при запуске программы из командной строки. Кроме того, в командной строке задается имя файла, в котором размещается фрагмент данных для поиска его в исходном файле. Программа «просматривает» исходный файл и выдает результат сравнения. В случае, если заданный фрагмент находится, то выдается соответствующее сообщение.

Затем исходный файл отправляется на стандартный вывод функцией _tprintf.

Листинг программы

```
CHAR string1[rmaxf1], string2[rmaxf2];
CHAR Buffer1[BUF_SIZE], Buffer2[BUF_SIZE];
DWORD FsLow, nf2;
int i, j, k, KEY;
BOOL f2;
LPCTSTR pF;
hFile = CreateFile (argv[1], GENERICREAD, 0, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
FsLow = GetFileSize (hFile, NULL);
printf("Fs=%d\n",(int)FsLow);
hMap = CreateFileMapping (hFile, NULL, PAGE_READONLY,
0, 0, NULL);
```

```
        printf("%c",string1[j]);
        }
        printf("\n");
        getchar();
        KCompare(string1, string2, (int)KEY);
        i=i+KEY;
        }
        while (i<(int)FsLow-(int)KEY+1);
        k=k+1;
        }
        while (k<(int)FsLow-(int)KEY+1);
/* Выводим исходный файл. */
_tprintf(_T("%s"),pFile);
UnmapViewOfFile (pFile);
CloseHandle (hMap);
CloseHandle (hFile);
CloseHandle (hFile2);
return 0;
}
```

Задание.

1. Запустить Microsoft Visua Studio C++ .NET 2003.

2. Создать проект консольного приложения под именем **lab4**.

3. Набрать текст программы отображения файла в памяти и поиска в нем заданного шаблона.

4.Провести отладку программы, создать исполняемый файл.

5. Запустить программу из командной строки, задавая имя файла имя исходного файла и имя файла, в котором находится фрагмент для поиска.

Контрольные вопросы:

1.Какую функцию выполняет CreateFileMapping?

2.Какую функцию выполняет OpenFileMapping?

3.Какую функцию выполняет MapViewOfFile??

4.Поясните работу функции KeyCompare.

5.Как запускается программа использования файла, отображенного в памяти?

Лабораторная работа №22

Тема: Использование динамических библиотек для создания приложений

Цель работы: *сформировать навыки работы с файлами.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Поток — это общее название, как ни странно, потока данных. В C++ поток представляет собой объект некоторого класса. Именно поэтому вы могли встретить в листингах потоковые объекты cin и cout.

Преимущества потоков

Программисты на C могут удивиться, какие же преимущества дает использование потоковых классов для ввода/вывода вместо традиционных функций C printf(), scanf(), для файлов — fprintf(), fscanf()...

Одним из аргументов в пользу потоков является простота использования. Если вам приходилось когда-нибудь использовать символ управления форматом %d при форматировании вывода с помощью %f в printf(), вы оцените это. Ничего подобного в потоках вы не встретите, ибо каждый объект сам знает, как он должен выглядеть на экране. Это избавляет программиста от одного из основных источников ошибок.

Файл – это именованная область ячеек памяти, в которой хранятся данные одного типа. Файл имеет следующие характерные особенности:

- уникальное имя;
- однотипность данных;
- произвольная длина, которая ограничивается только емкостью диска.

Файлы бывают текстовыми и двоичными.

Текстовый файл – файл, в котором каждый символ из используемого набора хранится в виде одного байта (код, соответствующий символу). Текстовые файлы разбиваются на несколько строк с помощью специального символа "конец строки". Текстовый файл заканчивается специальным символом "конец файла".

Двоичный файл – файл, данные которого представлены в бинарном виде. При записи в двоичный файл символы и числа записываются в виде последовательности байт (в своем внутреннем двоичном представлении в памяти компьютера).

Все операции ввода-вывода реализуются с помощью функций, которые находятся в библиотеке C++. Библиотека C++ поддерживает три уровня ввода-вывода:

потоковый ввод-вывод;

ввод-вывод нижнего уровня;

ввод-вывод для консоли и портов (зависит от ОС).

Поток – это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику.

Функции библиотеки ввода-вывода языка C++, поддерживающие обмен данными с файлами на уровне потока, позволяют обрабатывать данные различных размеров и форматов, обеспечивая при этом буферизованный ввод и вывод. Таким образом, поток представляет собой этот файл вместе с предоставленными средствами буферизации.

Чтение данных из потока называется **извлечением**, вывод в поток – **помещением** (включением).

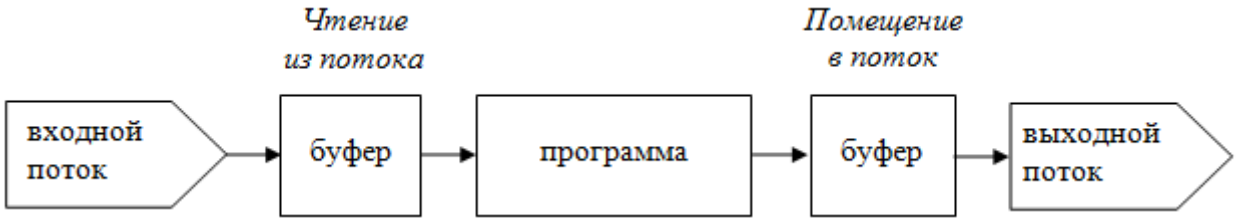


Рис. 1 Буферизация данных при работе с потоками

Для работы с файлом в языке C++ необходима ссылка на файл. Для определения такой ссылки существует структура **FILE**, описанная в заголовочном файле **stdio.h**. Данная структура содержит все необходимые поля для управления файлами, например: текущий указатель буфера, текущий счетчик байтов, базовый адрес буфера ввода-вывода, номер файла.

Функция открытия файла

При открытии файла (потока) в программу возвращается указатель на поток (файловый указатель), являющийся указателем на объект структурного типа FILE. Этот указатель идентифицирует поток во всех последующих операциях.

Например:

```
#include<stdio.h>
.............
FILE *fp;
```

Для открытия файла существует функция **fopen**, которая инициализирует файл.

Синтаксис:
fp=fopen(ИмяФайла, РежимОткрытия);
где **fp** – указатель на поток (файловый указатель);

ИмяФайла – указатель на строку символов, представляющую собой допустимое имя файла, в которое может входить спецификация файла (включает обозначение логического устройства, путь к файлу и собственно имя файла);

РежимОткрытия – указатель на строку режима открытия файла.

Режимы открытия файлов		
Режим	**Описание**	**Начина с...**
r	Открывает текстовый файл для чтения. Если файл не существует, то выдается ошибка при исполнении программы.	начала
w	Открывает текстовый файл для записи. Если файл не существует, то он создается. Если файл уже существует, то удаляется его содержимое, файл перезаписывается	начала
a	Открывает текстовый файл для добавления. Если файл не существует, то он создается. Если существует, то содержимое из него не удаляется.	конца
r+	Открывает текстовый файл для чтения и записи. Изменить размер файла нельзя. Если файл не существует, то выдается ошибка при исполнении программы.	начала
w+	Открывает текстовый файл для чтения и записи. Если файл не существует, то он создается. Если файл уже существует, то удаляется его содержимое, файл перезаписывается.	начала
a+	Открывает текстовый файл для чтения и записи. Если файл не существует, то он создается. Если существует, то содержимое из него не удаляется.	конца
rb	Открывает двоичный файл для чтения. Если файл не существует, то выдается ошибка при исполнении программы.	начала
wb	Открывает двоичный файл для записи. Если файл не существует, то он создается. Если файл уже существует, то удаляется его содержимое, файл перезаписывается.	начала
ab	Открывает двоичный файл для добавления. Если файл не существует, то он создается. Если существует, то содержимое из него не удаляется.	конца
r+b rb+	Открывает двоичный файл для чтения и записи. Изменить размер файла нельзя. Если файл не существует, то выдается ошибка при исполнении программы.	начала
w+b wb+	Открывает двоичный файл для чтения и записи. Если файл не существует, то он создается. Если файл уже существует, то удаляется его содержимое, файл перезаписывается.	начала
a+b ab+	Открывает двоичный файл для чтения и записи. Если файл не существует, то он создается. Если существует, то содержимое из него не удаляется.	конца

Например:

```
fp=fopen("t.txt","r");
```

Существуют несколько режимов открытия файлов.

Поток можно открыть в текстовом (t) или двоичном (b) режиме. По умолчанию используется текстовый режим. В явном виде режим указывается следующим образом:

"r+b" или "rb" – двоичный (бинарный) режим;

"r+t" или "rt" – текстовый режим.

Функция закрытия файла

Открытые на диске файлы после окончания работы с ними рекомендуется закрыть явно. Это является хорошим тоном в программировании.

Синтаксис:

int fclose(УказательНаПоток);

Возвращает 0 при успешном закрытии файла и -1 в противном случае.

Открытый файл можно открыть повторно (например, для изменения режима работы с ним) только после того, как файл будет закрыт с помощью функции fclose().

Функция удаления файла

Синтаксис:

int remove(const char *filename);

Эта функция удаляет с диска файл, указатель на который хранится в файловой переменной filename. Функция возвращает ненулевое значение, если файл невозможно удалить.

Функция переименования файла

Синтаксис:

int rename(const char *oldfilename, const char *newfilename);

Функция переименовывает файл; первый параметр – старое имя файла, второй – новое. Возвращает 0 при неудачном выполнении.

Функция контроля конца файла

Для контроля достижения конца файла есть функция feof.

int feof(FILE * filename);

Функция возвращает ненулевое значение, если достигнут конец файла.

Функции ввода-вывода данных файла

1) Символьный ввод-вывод

Для символьного ввода-вывода используются функции:

int fgetc(FILE *fp);

где **fp** – указатель на поток, из которого выполняется считывание.

Функция возвращает очередной символ в формате int из потока fp. Если символ не может быть прочитан, то возвращается значение EOF.

int fputc(int c, FILE*fp);

где **fp** – указатель на поток, в который выполняется запись;

c – переменная типа int, в которой содержится записываемый в поток символ.

Функция возвращает записанный в поток fp символ в формате int. Если символ не может быть записан, то возвращается значение EOF.

Пример 1.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR*
argv[]){
  FILE *f;
  int c;
  char *filename="t.txt";
  if ((f=fopen(filename,"r"))==0)
```

```
    perror(filename);
  else
    while((c = fgetc(f)) != EOF)
      putchar(c);
      //вывод с на стандартное
устройство вывода
  fclose(f);
  system("pause");
  return 0;
}
```

2) Строковый ввод-вывод

Для построчного ввода-вывода используются следующие функции:
char *fgets(char *s, int n, FILE *f);

где char *s – адрес, по которому размещаются считанные байты;

int n – количество считанных байтов;

FILE *f – указатель на файл, из которого производится считывание.

Прием байтов заканчивается после передачи n-1 байтов или при получении управляющего символа '\n'. Управляющий символ тоже передается в принимающую строку. Строка в любом случае заканчивается '\0'. При успешном завершении считывания функция возвращает указатель на прочитанную строку, при неуспешном – 0.
int fputs(char *s, FILE *f);

где char *s – адрес, из которого берутся записываемые в файл байты;

FILE *f – указатель на файл, в который производится запись.

Символ конца строки ('\0') в файл не записывается. Функция возвращает EOF, если при записи в файл произошла ошибка, при успешной записи возвращает неотрицательное число.

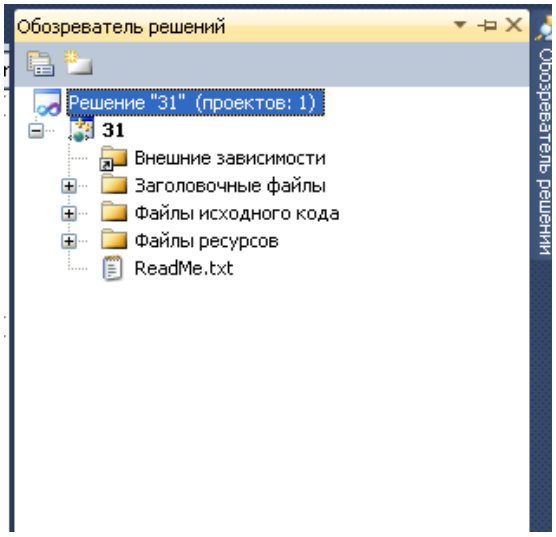
Пример 2. Построчное копирование данных из файла f1.txt в файл f2.txt.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
#define MAXLINE 255 //максимальная длина строки
int _tmain(int argc, _TCHAR* argv[]){
//копирование файла in в файл out
  FILE *in, //исходный файл
       *out; //принимающий файл
  char buf[MAXLINE];
  //строка, с помощью которой выполняется
копирование
  in=fopen("f1.txt","r");
  //открыть исходный файл для чтения
  out=fopen("f2.txt","w");
  //открыть принимающий файл для записи
  while(fgets(buf, MAXLINE, in)!=0)
  //прочитать байты из файла in в строку buf
    fputs(buf, out);
    //записать байты из строки buf в файл out
  fclose(in); //закрыть исходный файл
  fclose(out);//закрыть принимающий файл
  system("pause");
  return 0;
}
```

Замечание: отобразите окно свойств для элементов (***Вид→Другие окна→Окно свойств*** или ***Alt+Enter***).

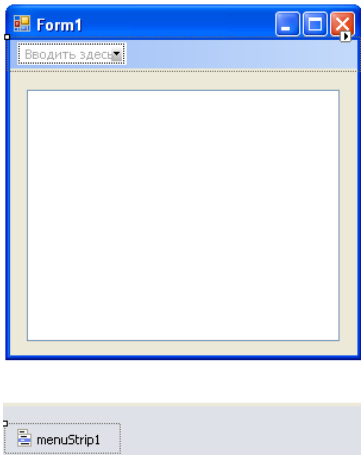
Замечание: знакомьтесь с назначением элементов формы, просматривая всплывающие подсказки:

Стандартное содержимое вашего проекта можно увидеть в Обозревателе решений:



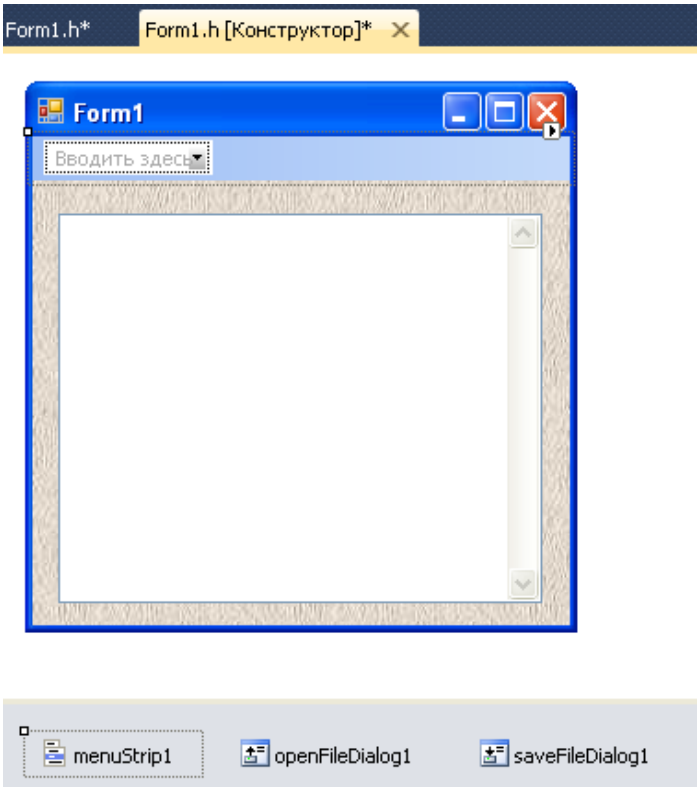
1. Создайте новый проект C++
(*Файл→Создать→Проект→ВыбратьCLR+ "Приложение Windows Forms"*).
2. Разместите на форме:
    a. элемент **MenuStrip;**
    b. элемент **openFiledialog**;
    c. элемент **saveFiledialog;**
    d. элемент **textBox1**.

Примерный вид формы с необходимыми элементами:



3. Измените свойства для следующих элементов:
    – для **Form1** – Text= Создание меню;
    – для **textBox1** – Multeline, ScrollBars=Vertical.

Примерный вид формы:

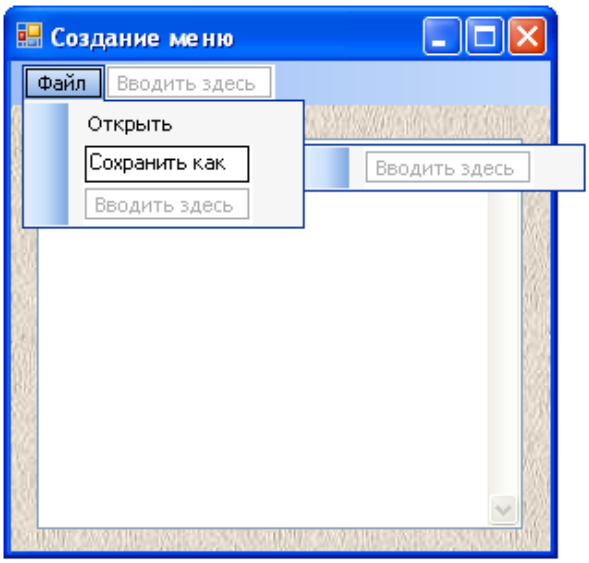


4. Определите поле Файл для меню:
    – кликните по текстовому полю "Вводить здесь";

– введите имя пункта – "Файл”.

5. Добавьте остальные пункты:

   – в нижнем новом появившемся текстовом поле напишите "Открыть";
   – аналогично добавьте "Сохранить как".



6. Определите возможность **textBox** изменяться пропорционально размеру формы при увеличении окна (*по умолчанию **textBox** “привязан” к левой и верхней части формы*).

   Для этого задайте свойство **Achor** для **textBox1.** Выделите правую и нижнюю часть, после чего размер элемента **textBox** будет изменяться как надо.

7. Выполните отладку (F5) и проверьте работоспособность. Ширина и высота **textBox** меняется автоматически.

8. Внимательно построчно разберите нижеприведённый код. Напишите код обработки событий для добавленных элементов:

```
#pragma endregion


private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e) {
             this->Text = "Создание меню";
 openFileDialog1->FileName = "C:\\Documents and Settings\\user216.AUD\\Рабочий стол\\41.txt";
 openFileDialog1->Filter = "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
 saveFileDialog1->Filter = "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
         }

private: System::Void открытьToolStripMenuItem_Click(System::Object^  sender, System::EventArgs^  e) {

            openFileDialog1->ShowDialog();
 if (openFileDialog1->FileName == nullptr) return;
 try
 {
 auto MyReader = gcnew IO::StreamReader(openFileDialog1->FileName, System::Text::Encoding::GetEncoding(1251));
 textBox1->Text= MyReader->ReadToEnd();
 MyReader->Close();
 }
 catch (IO::FileNotFoundException^ Ситуация)
 {
 MessageBox::Show(Ситуация->Message + "\nФайл не найден", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Exclamati
 }
 catch (Exception^ Ситуация)
 {
 MessageBox::Show(Ситуация->Message, "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
 }
         }
```

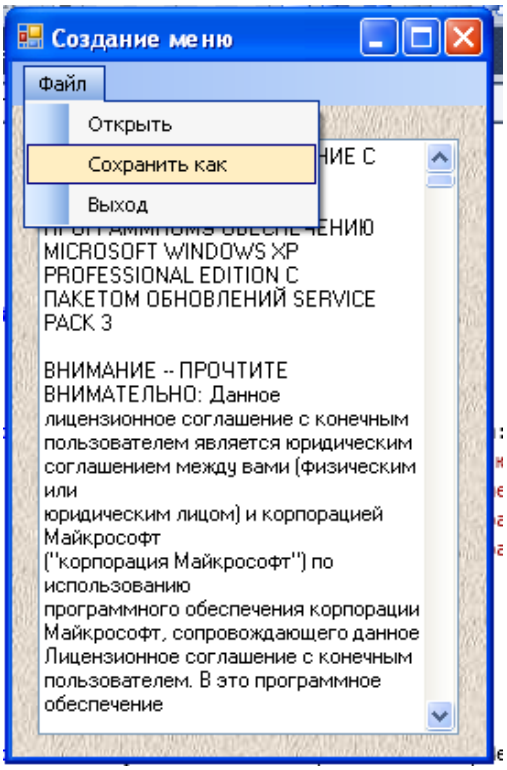
orm1

```
private: System::Void сохранитьКакToolStripMenuItem_Click(System::Object^  sender, System::EventArgs^  e) {

            saveFileDialog1->FileName = openFileDialog1->FileName;
 if (saveFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) Save();
 }
 void Save()
 {
 try
 {
 // Создание экземпляра StreamWriter для записи в файл:
 auto MyWriter = gcnew IO::StreamWriter(saveFileDialog1->FileName, false, System::Text::Encoding::GetEncoding(1251))
 MyWriter->Write(textBox1->Text);
 MyWriter->Close(); textBox1->Modified = false;
 }
 catch (Exception^ Ситуация)
 {
 MessageBox::Show(Ситуация->Message, "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
 }
         }



        private: System::Void Form1_FormClosing(System::Object^ sender, System::Windows::Forms::FormClosingEventArg
 if (textBox1->Modified == false) return;
 auto MeBox = MessageBox::Show("Текст был изменён. \nСохранить изменения?",
 "Простой редактор", MessageBoxButtons::YesNoCancel, MessageBoxIcon::Exclamation);
 if (MeBox == Windows::Forms::DialogResult::No) return;
 if (MeBox == Windows::Forms::DialogResult::Cancel) e->Cancel = true;
 if (MeBox == Windows::Forms::DialogResult::Yes)
 {
 if (saveFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK)
 {
 Save(); return;
 }
 else e->Cancel = true;
 }
 }
```

9. Добавьте пункт меню Файл – Выход, код для него: this->Close();

10. Выполните отладку (F5) и проверьте работоспособность.

Примерный результат работы вашей программы:



11. Покажите выполненную работу преподавателю и получите дополнительное задание.

12. Ответьте на контрольные вопросы.

Контрольные вопросы:

1. Какова функция элемента **MenuStrip**?
2. Что определяет свойство Vertical элемента формы **textBox**?
3. Как создать элемент формы “вкладка”?
4. Что определяет свойство FileName элемента формы openFileDialog1?

Лабораторная работа №23

Тема: Многопроцессная обработка данных.

Цель работы: *сформировать навыки работы с цепочечными командами.*

Ход работы:

1. Выполнить задание в соответствии с указаниями
2. Ответить на контрольные вопросы
3. Предъявить преподавателю результаты работы: проект и исходный код
4. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Кроме привычного всем понятия массивов в ассемблере существует структура называемая цепочкой. Цепочка - непрерывная последовательность байт, слов или двойных слов, обрабатываемая как единое целое. Основное отличие цепочек от массивов состоит в способе доступа к элементам: для массивов - произвольный доступ, для цепочек - только

последовательный (от начала цепочки к концу или от конца к началу). Цепочечные команды - команды для обработки цепочек. Особенностью всех цепочечных команд (кроме обработки очередного элемента цепочки) является автоматическое продвижение к следующему элементу цепочки.

Цепочечные команды:

Название	**Команды**	**Действие**
пересылка цепочки	movs <адр. приемника>, <адр. источника> movsb, movsw, mowsd	копирует один элемент цепочки из операнда источника в операнд приемник
сравнение цепочек	cmps <адр. приемника>, <адр. источника> cmpsb, cmpsw, cmpsd	сравнивает элементы цепочек из операнда источника и операнда приемника
сканирование цепочки	scas <адр. приемника> scasb, scasw, scasd	сканирует цепочку приёмник на присутствие некоторого элемента (задаётся в регистре аккумуляторе)
загрузка элемента из цепочки	lods <адр. источника> lodsb, lodsw, lodsd	загрузить элемент из цепочки источника в регистр аккумулятор
сохранение элемента в цепочке	stos <адр. приемника> stosb, stosw, stosd	восстановить элемент из регистра аккумулятора в цепочку
получение элемента цепочки из порта ввода/вывода	ins <адр. приемника>, <номер порта> insb, insw, insd	загрузить элемент в цепочку приемник из указанного порта ввода/вывода
вывод элементов цепочки в порт ввода/вывода	outs <номер порта>, <адр. источника> outbs, outws, outds	переслать элемент из цепочки источника в указанный порт ввода/вывода

ОСОБЕННОСТИ ИСПОЛЬЗОВАНИЯ

Адресация операндов

цепочка источник - *ds:si*

цепочка приёмник - *es:di*

Направление обработки

1. от начала к концу
   df = 0; *si* и *di* автоматически увеличиваются
   команда cld (clear direction flag) сбрасывает флаг df
2. от конца к началу
   df = 1; *si* и *di* автоматически уменьшаются
   команда std (set direction flag) устанавливает флаг df

Префиксы повторения

Префиксы повторения зацикливают выполнение команды, позволяя обрабатывать всю цепочку одной командой. Префиксы повторения указываются перед нужной цепочечной командой в поле метки. Цепочечная команда без префикса повторения выполняется один раз, с префиксом - в цикле.

rep выполнять, пока cx<>0 (cx уменьшается автоматически);
repe выполнять, пока cx<>0 или zf=1 (cx уменьшается автоматически);
repne выполнять, пока cx<>0 или zf=0 (cx уменьшается автоматически);

Пример: Написать программу копирования строки.

```
1   data segment
2       s1 db 'Тестируемая строка$' ;строка которую будем копировать
3       s2 db 20 dup (' ')          ;строка куда будем копировать
4   data ends
5
6   code segment
7   start:
8       assume cs:code, ds: data
9       mov ax, data
10      mov ds, ax                  ;цепочка источник
11      mov es, ax                  ;и цепочка приёмник в одном сегменте
12
13      cld                         ;обработка от начала к концу
14      lea si, s1                  ;цепочка источник
15      lea di, s2                  ;цепочка приёмник
16      mov cx, 20                  ;количество элементов для обработки
17  rep movsb                       ;копируем строку
18
19      mov ah, 09
20      lea dx, s2
21      int 21h                     ;выводим строку-приёмник на экран
22
23      mov ax, 4c00h
24      int 21h
25  code ends
26  end start
```

Задание 1. Write(S). Вывести строку S на экран (конец строки задается символом с кодом 0).

Задание 2. Length(S). Определить длину строки S (конец строки задается символом с кодом 0).

Задание 3. Copy(S, p, i). Скопировать из строки S подстроку длиной i символов начиная с позиции p.

Контрольные вопросы:

1. Что такое массивы и в каких случаях они используются?
2. Как организуется доступ к элементам массивов?
3. Какие еще структурированные типы данных поддерживаются языками высокого уровня?

Лабораторная работа №24

Тема: Расширенный ввод-вывод с процедурами завершения.

Цель работы: *сформировать навыки организации ввода-вывода с процедурами завершения.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Использованию объектов синхронизации есть альтернатива, которая заключается в том, что вместо ожидания потоком сигнала завершения от события или дескриптора, система может вызвать указанную пользователем процедуру завершения, когда операция ввода-вывода закончится. Эта процедура завершения может запустить следующую операцию ввода-вывода и выполнить любое другое действие.

Как программа может задать процедуру завершения?

В ReadFile или WriteFile не остается никаких параметров или структур данных, в которых можно было бы сохранить адрес процедуры. Однако существует семейство расширенных функций ввода-вывода, отличающихся суффиксом Ex, которые имеют дополнительный параметр для адреса процедуры завершения.

Расширенные функции чтения и записи - ReadFileEx и WriteFileEx.

Кроме того, следует использовать одну из пяти ожидающих функций предупреждения.

- WaitForSingleObjectEx
- WaitForMultipleObjectsEx
- SleepEx
- SignalObjectAndWait
- MsgWaitForMultipleObjectsEx

Расширенный ввод-вывод иногда называется вводом-выводом с предупреждением

Расширенные функции чтения и записи можно применять с дескрипторами открытых файлов, именованных каналов и почтовых ячеек, если при открытии (создании) был указан флаг FILE_FLAG_OVERLAPPED. Хотя ввод-вывод с перекрытием и расширенный ввод-вывод отличаются друг от друга, этот флаг устанавливает атрибут дескриптора для обоих типов асинхронного ввода-вывода.

1.Функции ReadFileEx и WriteFileEx

```
BOOL ReadFileEx (
    HANDLE hFile,
    LPVOID lpBuffer
    DWORD  nNumberOfBytesToRead,
    LPOVERLAPPED lpOverlapped,.

    LPOVERLAPPED_COMPLETION_ROUTINE lpcr);

BOOL WriteFileEx  (
    HANDLE hFile,
    LPVOID lpBuffer,
    DWORD nNumberOfBytesToWrite,
    LPOVERLAPPED lpOverlapped,
    LPOVERLAPPED_COMPLETION_ROUTINE lpcr);
```

Эти две функции содержат дополнительный параметр для указания процедуры завершения.

Структуры перекрытия можно задать, но указывать в них элемент hEvent нет необходимости, так как система его проигнорирует. В то же время этот элемент может принести пользу, если фиксировать в нем данные идентификации.

В отличие от ReadFile и WriteFile, расширенные функции не требуют параметров, определяющих количество передаваемых байтов. Эта информация передается в процедуру завершения, которая должна быть включена в программу.

Процедура завершения принимает параметры количества байтов, кода ошибки и структуры перекрытия. Последний параметр нужен для того, чтобы процедура могла определить, какая из нескольких незаконченных операций завершается. В этом случае надо учитывать такие же ограничения по повторному использованию или уничтожению структур, как и при вводе-выводе с перекрытием.

```
VOID WINAPI  FileIOCompletionRoutine (
    DWORD fdwError,
    DWORD cbTransferred,
    LPOVERLAPPED lpo);
```

Как и в случае с CreateThread, где также указывалось имя функции, здесь FileIOCompletionRoutine- метка-заполнитель, а не фактическое имя функции.

fdwError может принимать лишь значения 0 (успех) и ERROR_HANDLE_EOF (при попытке чтения после конца файла).

Структура перекрытия используется из завершенного вызова ReadFileEx или WriteFileEx.

Прежде чем система вызовет процедуру завершения, должно произойти следующее.

1. Операция ввода-вывода должна завершиться.

2. Вызывающий поток должен находиться в состоянии ожидания предупреждения, которое информирует систему о том, что она должна выполнить процедуру завершения из имеющейся очереди.

Чтобы перейти в состояние ожидания утверждения, поток должен сделать явный вызов одной из трех ожидающих функций предупреждения. Таким образом, поток гарантирует, что процедура завершения не будет вызвана преждевременно.

Когда эти два условия выполняются, запускаются процедуры завершения, которые были помещены в очередь как результат завершения ввода-вывода. Процедуры завершения выполняются в том же потоке, который вызвал первоначальный ввод-вывод и который находится в состоянии ожидания предупреждения. Поэтому поток должен переходить в это состояние только в том случае, если выполнение процедур завершения не приведет к нежелательным результатам

2.Ожидающие функции предупреждения

Существует пять ожидающих функций предупреждения, три из которых имеют непосредственное отношение к нашему изложению и описаны здесь. Каждая из них имеет флаг fAlertable, который должен быть установлен в TRUE. Эти функции являются расширением обычных функций Wait и Sleep.

```
DWORD  WaitForSingleObjectEx  (
     HANDLE  hObject,
     DWORD dwTimeOut,
     BOOL fAlertable);

DWORD                WaitforMultipleObjectEx                    (
DWORD  cObjects,
     LPHANDLE  lphObjects,
     BOOL   fWaitAll,
     DWORD dwTimeOut,
     BOOL fAlertable);

DWORD SleepEx (
     DWORD dwTimeOut,
     BOOL fAlertable);
```

Тайм-ауты, как обычно, задаются в миллисекундах. Эти три функции возвращают управление, как только наступает любая из следующих ситуаций:

• дескриптор или дескрипторы вызывают удовлетворение одной из двух функций ожидания обычным образом;

• истекает тайм-аут;

• все процедуры завершения в очереди потока завершаются, и флаг fAlertable установлен.

Со структурами перекрытия ReadFileEx и WriteFileEx не связано никаких событий, так что дескрипторы в вызове функции ожидания не будут иметь прямого отношения к операциям ввода-вывода. Функция SleepEx, с другой стороны, не связана с объектом синхронизации и наиболее проста в использовании. В этой функции обычно указывается значение тайм-аута INFINITE, в результате чего она возвращает управление только по окончании всех процедур завершения в очереди.

3.Программа преобразование файла с использованием расширенного ввода-вывода

Программа lab7.exe обеспечивает преобразование файла ASCII в Unicode с использованием асинхронного ввода-вывода с помощью процедуры завершения. Многие переменные сделаны глобальными, для того чтобы они были доступны для процедур завершения. Запуск программы производится из командной строки в виде: lab7.exe filel file2.

```
#include "stdafx.h"
#include "EvryThng.h"
#define MAX_OVRLP 4
#define REC_SIZE 64
#define UREC_SIZE 2 * REC_SIZE

static VOID WINAPI ReadDone (DWORD, DWORD, LPOVERLAPPED);
static VOID WINAPI WriteDone (DWORD, DWORD, LPOVERLAPPED);
/* Первая структура перекрытия предназначена для чтения, а вторая для
записи. Структуры и буфера выделяются для каждой незавершенной
операции. */
```

```
OVERLAPPED OverLapIn [MAX_OVRLP],OverLapOut [MAX_OVRLP];
CHAR AsRec [MAX_OVRLP][REC_SIZE];
WCHAR UnRec [MAX_OVRLP] [REC_SIZE];
HANDLE hInputFile, hOutputFile;
LONGLONG nRecord, nDone;
LARGE_INTEGER FileSize;
LARGE_INTEGER CurPosIn, CurPosOut;
DWORD ic;

int _tmain (int argc, LPTSTR argv [])
{
hInputFile = CreateFile (argv [1], GENERIC_READ,
0, NULL, OPEN_EXISTING, FILE_FLAG_OVERLAPPED, NULL);

hOutputFile = CreateFile (argv [2], GENERIC_WRITE,
0, NULL, CREATE ALWAYS, FILE_FLAG_OVERLAPPED, NULL);
FileSize.LowPart = GetFileSize (hInputFile,
(LPDWORD)FileSize.HighPart);
Record = FileSize.QuadPart / REC_SIZE;
printf("nR=%d\n", (int)nRecord);
if ((FileSize.QuadPart % REC_SIZE) != 0) nRecord++;
CurPosIn.QuadPart =0;
printf("nR2=%d\n", (int)nRecord);
for (ic = 0; ic < MAX_OVRLP; ic++)
{
printf("ic=%d\n", (int)ic);
OverLapIn [ic].hEvent = (HANDLE) ic; /* Перезагрузка события. */
OverLapOut [ic].hEvent = (HANDLE) ic;
OverLapIn [ic].Offset = CurPosIn.LowPart;
OverLapIn [ic].OffsetHigh = CurPosIn.HighPart;
if (CurPosIn.QuadPart < FileSize.QuadPart)
ReadFileEx (hInputFile, AsRec [ic], REC_SIZE, &OverLapIn [ic],
ReadDone);
CurPosIn.QuadPart += (LONGLONG) REC SIZE;
}
 /* Все операции чтения выполняются. Вводим состояние ожидания
завершения и продолжаем, пока не будут обработаны все записи. */
nDone = 0;
while (nDone < 2 * nRecord)
SleepEx (ININITE, TRUE);
CloseHandle (InputFile);
CloseHandle (hOutputFile);
_tprintf (_T ("ASCII in Unicode is completed\n"));
return 0;
}
static VOID WINAPI ReadDone (DWORD Code, DWORD nBytes,
LPOVERLAPPED pOv)
{
/* Чтение завершено. Преобразуем данные и начинаем запись. */
DWORD i;
Done++;
printf("nD1=%d\n", (int)nDone);
/* Обработка записи и начало операции записи. */
ic = (DWORD) (pOv->hEvent);
CurPosIn.LowPart = OverLapIn [ic].Offset;
```

```
CurPosIn.HighPart = OverLapIn [ic].OffsetHigh;
CurPosOut.QuadPart = (CurPosIn.QuadPart / REC_SIZE) * UREC_SIZE;
OverLapOut [ic].Offset = CurPosOut.LowPart;
OverLapOut [ic].OffsetHigh = CurPosOut.HighPart;
/* Преобразование записи ASCII в Unicode. */
for (i = 0; i < nBytes; i++)
UnRec [ic] [i] = AsRec [ic] [i];
WriteFileEx (hOutputFile, UnRec [ic], nBytes*2,
 &OverLapOut [ ic ], WriteDone);
/* Подготовка структуры перекрытия к следующему чтению. */
CurPosIn.QuadPart += REC_SIZE * (LONGLONG) (MAX_OVRLP);
OverLapIn [ic].Offset = CurPosIn.LowPart;
OverLapIn [ic].OffsetHigh = CurPosIn.HighPart;
return;

static VOID WINAPI WriteDone (DWORD Code, DWORD nBytes,
LPOVERLAPPED pOv)
{
/* Запись завершена. Начинаем следующее чтение. */
/*LARGE_INTEGER CurPosIn;*/
/*DWORD ic;*/
nDone++;
printf("nD2=%d\n", (int)nDone);
ic = (DWORD) (pOv->hEvent);
CurPosIn.LowPart = OverLapIn [ic].Offset;
CurPosIn.HighPart = OverLapIn [ic].OffsetHigh;
if (CurPosIn.QuadPart < FileSize.QuadPart)
{
ReadFileEx (hInputFile, AsRec [ic], REC_SIZE, &OverLapIn [ic],
ReadDone);
}
return;
}
```

Задание 1.

1. Запустить Microsoft Visual Studio C++ /NET 2003.
2. Создать проект в консольном приложении под именем lab7.
3. В файл lab7.cpp набрать текст программы.
4. Провести отладку программы, создать исполняемый файл.
5. Запустить программу из командной строки, задавая имя исходного файла в кодировке ASCII и имя файла, в который помещается преобразованный файл.

Контрольные вопросы:

1. Какую фунцию выполняетReadFileEx?

2.Какую фунцию выполняетWriteFileEx?

3.Какую фунцию выполняет SleepEx?

4.Какую фунцию выполняет WaitForSingleObjectEx?

5.Поясните работу процедуры WINAPI ReadDone?

6.Поясните работу процедуры WINAPI WriteDone?

Лабораторная работа №25

Тема: Рисование графических фигур на экране монитора.

Цель работы: *сформировать навыки работы с командами построения элементарных графических фигур.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

1.Библиотека классов Microsoft Foundation Classes (MFC)

Библиотека классов Microsoft Foundation Classes (MFC) содержит набор средств объектно-ориентированного программирования, предназначенных для разработки 32-разрядных приложений. MFC – это мощный инструментарий объектно-ориентированного программирования.

Библиотека MFC дает возможность использования различных классов объектов. Наиболее важные структуры данных и API-функции инкапсулированы в группы классов, пригодных для многократного использования.

По сравнению с традиционными библиотеками функций, которые использовались в программах на языке С, библиотека MFC имеет много достоинств.

Основные преимущества использования классов C++ следующие:

• устраняются конфликты, возникшие из-за совпадения имен функций и переменных;

• код и данные инкапсулируются в классах;

• осуществляется наследование функциональных возможностей;

• размер кода значительно сокращается за счет хорошо организованной библиотеки классов;

• создаваемые программистом классы являются естественным расширением языка.

Благодаря MFC код, требуемый для отображения окна приложения, можно сократить примерно в три раза. При этом появляется возможность уделять больше времени и внимания разработке процедур взаимодействия приложения с Windows, для решения которых и создается приложение.

В основу MFC положены следующие принципы:

• возможность комбинирования обычных вызовов функций с методами классов;

• баланс между производительностью и эффективностью базовых классов;

• преемственность в переходе от использования API-функций к библиотеке классов;

• простоту переноса библиотеки классов на разные платформы ОС.

• наиболее полное использование возможностей C++ без чрезмерного усложнения программного кода.

Простота базовых классов делает их весьма несложными в использовании, а по скорости работы их методы практически не уступают библиотечным функциям языка С.

При создании MFC также учтены возможности работы в смешанном режиме. Другими словами, в одной программе могут применяться как библиотека MFC, так и API-функции.

Еще одно важное свойство, MFC заключается в том, что имеется возможность непосредственного применения базовых классов в программах.

Основные достоинства библиотеки MFC:

• Расширенная система обработки исключительных ситуаций, благодаря которой приложения менее чувствительны к ошибкам и сбоям. Такие ошибки, как «нехватка памяти» обрабатываются автоматически.

• Улучшенная система диагностики, позволяющая записывать в файл информацию об используемых объектах. Сюда также следует отнести возможность контроля корректности значений переменных - членов.

• Полная поддержка всех API-функций, элементов управления, сообщений, GDI, графических примитивов, меню и диалоговых окон.

• Возможность определить тип объекта во время выполнения программы. Это позволяет осуществлять динамическое манипулирование свойствами экземпляров классов.

• Отпала необходимость в использовании многочисленных громоздких структур switch/case, которые часто являются источниками ошибок. Все сообщения связываются с их обработчиками внутри классов. Этот способ привязки сообщений к обработчикам событий применим ко всем сообщениям.

• Небольшой размер и быстрота выполнения программного кода. Как уже говорилось выше, по этим показателям MFC не уступает стандартным библиотечным функциям языка С.

• Поддержка COM (Component Object Model – модель компонентных объектов).

• Использование тех же соглашений об именовании методов классов, которые применялись при подборе имен для API-функций Windows. Это существенно облегчает идентификацию действий, выполняемых классами.

Библиотека MFC поставляется вместе с исходными текстами классов, что дает возможность настраивать базовые классы в соответствии со своими потребностями.

2.Создание проекта программы рисования фигур

Используя мастер Application Wizard, создадим проект с именем GDI. Для этого необходимо:

• В меню File открыть вкладку New, Project, выбрать иконку MFC Application, в окне Name: набрать имя gdi, в окне Location: указать путь к папке, в которую будет занисан проект, нажать ОК.

• На вкладке MFC Application Wizard – выбрать Application Type и установить Single documents (поддержка единичного документа), MFC standard, Use MFC in a static library, Document/View architecture support, Resource language – Английский (США).

• На вкладке MFC Application Wizard – выбрать Compound Document Support, установить None.

• На вкладке MFC Application Wizard - Document Template Strings должны стоять стандартные установки .

• На вкладке MFC Application Wizard – выбрать Database support и установить None.

• На вкладке MFC Application Wizard – выбрать User Interface Features и установить Thick frame, Minimize box, Maximize box, System menu, Toolbars - None.

• На вкладке MFC Application Wizard – выбрать Advance Features, снять все установки, Number of files on recent file list – 4.

• На вкладке MFC Application Wizard – Generated Classes должен появится перечень из 4 классов проекта: CgdiView, CgdiApp, CgdiDoc, CMainFrame.

• В итоге получим папку с файлами проекта Gdi. Если папка с файлами проекта не появилась на экране, необходимо выбрать в меню View, Solution Explorer. Открыв папку Source Files, можно убедиться в наличии файлов проекта.

Для того, чтобы иметь возможность рисовать в рабочей области приложения графические фигуры, нужно предварительно модифицировать часть файла исходного кода gdiView.cpp. Для этого необходимо открыть этот файл, щелкнув по нему мышью, затем в этом файле найти фрагмент, озаглавленный как CgdiView drawing.

Файл GdiView.cpp, который содержит реализацию класса CGdiView, порождается от класса CGdiView и управляет отображением документа на экране монитора. В файл класса CGdiView добавлен фрагмент, выделенный полужирным шрифтом, который позволяет производит окраску графических фигур указанными цветами.

3.Программа рисования фигур

```
// GdiView.cpp : реализация класса CGdiView
#include "stdafx.h"
#include "Gdi. h"
#include "GdiDoc.h"
#include "GdiView.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif

//CGdiViewdrawing
      void CgdiView :: OnDraw(CDC* pDC)
      {
        CGdiDoc* pDoc = GetDocument( );
        ASSERT_VALID(pDoc);
      static DWORD dwColor[9]={
        RGB(0,0,0),
        //черный
        RGB(255,0,0),                                    //красный
        RGB(0,255,0),                                    //зеленый
        RGB(0,0,255),                                    //синий
        RGB(255,255,0),                                  //желтый
        RGB(255,0,255),                                  //пурпурный
```

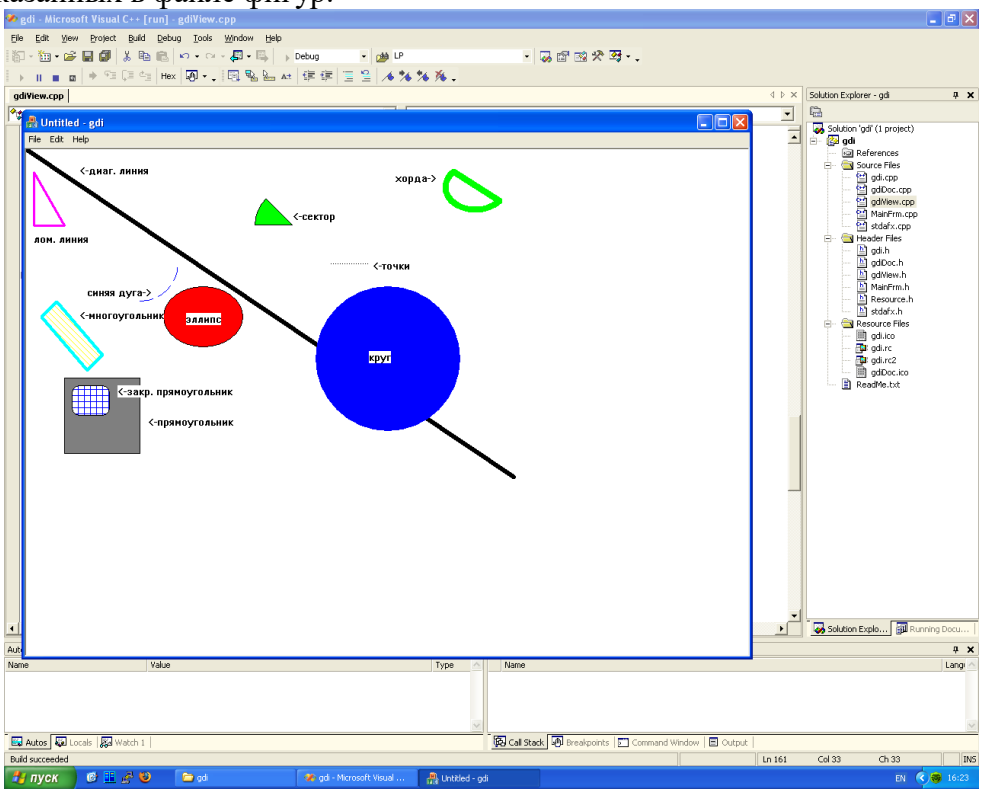
```
  RGB(0,255,255),                                   //голубой
  RGB(127,127,127),                                 //серый
  RGB(255,255,255)};                                //белый

int xcoord;
POINT polylpts[4],polygpts[5];
CBrush newbrush;
CBrush* oldbrush;
CPen newpen;
CPen* oldpen;
// рисование толстой диагональной линии
newpen.CreatePen(PS_SOLID,6,dwColor[0]);
oldpen=pDC->SelectObject(&newpen);
pDC->MoveTo(0,0);
pDC->LineTo(640,430);
pDC->TextOut(70,20,"<-diagonalline",15);
// удаление пера
pDC->SelectObject (oldpen);
newpen.DeleteObject();

// рисование синей дуги
newpen.CreatePen(PS_DASH,1,dwColor[3]);
oldpen=pDC->SelectObject(&newpen);
pDC->Arc(100,100,200,200,150,175,175,150);
pDC->TextOut(80,180,"small arc->",11);
// удаление пера
pDC->SelectObject(oldpen);
newpen.DeleteObject( );

// рисование сегмента с толстым зеленым контуром
newpen.CreatePen(PS_SOLID,8,dwColor[2]);
oldpen=pDC->SelectObject(&newpen);
pDC->Chord(550,20,630,80,555,25,625,70);
pDC->TextOut(485,30,"chord->",7);
// удаление пера
pDC->SelectObject(oldpen);
newpen.DeleteObject ( );

// рисование эллипса и заливка его красным цветом
CreatePen(PS_SOLID,1,dwColor[1]);
oldpen=pDC->SelectObject(&newpen);
```

```
newbrush.CreateSolidBrush(dwColor[1]);
oldbrush=pDC->SelectObject(&newbrush);
pDC->Ellipse(180,180,285,260);
pDC->TextOut(210,215, "ellipse", 7);
// удаление кисти
pDC->SelectObject(oldbrush);
newbrush.DeleteObject();
// удаление пера
pDC->SelectObject(oldpen);
newpen. DeleteObject ( );

// рисование  круга и заливка его синим цветом
newpen.CreatePen(PS_SOLID,1,dwColor[3]);
oldpen=pDC->SelectObject(&newpen);
newbrush.CreateSolidBrush (dwColor[3]);
oldbrush=pDC->SelectObject(&newbrush);
pDC->Ellipse(380,180,570,370);
pDC->TextOut(450,265,"circle",6);
// удаление кисти
pDC->SelectObject(oldbrush);
newbrush.DeleteObject() ;
// удаление пера
pDC->SelectObject(oldpen) ;
newpen.DeleteObject();

 // рисование сектора и заливка его зеленым цветом
newpen.CreatePen(PS_SOLID,1,dwColor[0]);
oldpen=pDC->SelectObject(&newpen);
newbrush.CreateSolidBrush(dwColor[2]);
oldbrush=pDC->SelectObject(&newbrush);
pDC->Pie(300,50,400,150,300,50,300,100);
pDC->TextOut(350,80,"<-pie wedge",11);
// удаление кисти
pDC->SelectObject(oldbrush) ;
newbrush.DeleteObject();
//удаление пера
pDC->SelectObject(oldpen) ;
newpen.DeleteObject();

// рисование прямоугольника и заливка его серым цветом
newbrush.CreateSolidBrush(dwColor[7]);
```

```
oldbrush=pDC->SelectObject(&newbrush);
pDC->Rectangle(50,300,150,400);
pDC->TextOut(160,350,"<-rectangle",11);
// удаление кисти
pDC->SelectObject(oldbrush);
newbrush.DeleteObject( );

// рисование закругленного прямоугольника
// и заливка его синим цветом
newbrush.CreateHatchBrush(HS_CROSS,dwColor[3]);
oldbrush=pDC->SelectObject(&newbrush);
pDC->RoundRect(60,310,110,350,20,20);
pDC->TextOut (120,310,"<-rounded rectangle" ,19);
// удаление кисти
pDC->SelectObject(oldbrush);
newbrush.DeleteObject( );

// рисование нескольких точеск
for(xcoord=400;xcoord<450;xcoord+=3)
pDC->SetPixel(xcoord,150,0L);
pDC->TextOut(455,145,"<-pixels",8);

// рисование толстой ломаной линии пурпурного цвета
newpen.CreatePen(PS_SOLID,3,dwColor[5]);
oldpen=pDC->SelectObject(&newpen);
polylpts[0].x=10;
polylpts[0].y=30;
polylpts[1].x=10;
polylpts[1].y=100;
polylpts[2].x=50;
polylpts[2].y=100;
polylpts[3].x=10;
polylpts[3].y=30;
pDC->Polyline(polylpts,4);
pDC->TextOut(10,110,"polyline",8);
// удаление пера
pDC->SelectObject(oldpen);
newpen.DeleteObject( );

// рисование многоугольника с голубым контуром
// и заливка его диагональной желтой штриховкой
```

```
        newpen.CreatePen(PS_SOLID,4,dwColor[6]);
        oldpen=pDC->SelectObject(&newpen);
        newbrush.CreateHatchBrush(HS_FDIAGONAL,dwColor[4]);
        oldbrush=pDC->SelectObject(&newbrush);
        polygpts[0].x=40;
        polygpts[0].y=200;
        polygpts[1].x=100;
        polygpts[1].y=270;
        polygpts[2].x=80;
        polygpts[2].y=290;
        polygpts[3].x=20;
        polygpts[3].y=220;
        polygpts[4].x=40;
        polygpts[4].y=200;
        pDC->Polygon(polygpts,5);
        pDC->TextOut(70,210,"<-polygon",9);
        // удаление кисти
        pDC->SelectObject(oldbrush);
        newbrush.DeleteObject ();
        // удаление пера
        pDC-> SelectObject(oldpen);
        newpen.DeleteObject( );
}
```

После ввода кода рисования фигур необходимо выполнить компиляцию программы, для этого в меню Build следует выбрать Build Solution. В результате компиляции в папке Debug проекта будет создан файл Cview.cpp, выполнение которого обеспечивает рисование указанных в файле фигур.



4.Пояснения к программе рисования фигур

Рисование линий и геометрических фигур с заданным цветом выполняется программируемыми «перьями», а раскраска фигур осуществляется «кистями», которые реализуются в классах CBrush и CPen.

Структура dwColor создается массив, в котором хранятся девять RGB-значений цветов для используемых кистей и перьев:

```
static DWORD dwColor[9]=
{
RGB(0,0,0),              //черный
RGB(255,0,0),            //красный
RGB(0,255,0),            //зеленый
RGBCO,0,255),            //синий
RGB(255,255,0),          //желтый

  RGB(255,0,255),          //пурпурный
  RGB(0,255,255),          //голубой
  RGB(127,127,127),        //серый
  RGB(255,255,255)         //белый
  };
```

Кисти и перья создаются с помощью классов CBrush и CPen, которые можно передавать функциям класса CDC - базового класса для контекста устройства отображения. Кисти могут иметь сплошную и штриховую заливку, а также заливку в виде растрового узора. Перья рисуют сплошной, штриховой или пунктирной линиями. Синтаксис программы, отвечающей за создание кистей перьев, приведен ниже:

```
CBrush newbrush;
CBrush* oldbrush;
CPen  newpen;
CPen* oldpen;
```

Поскольку для рисования различных графических примитивов применяются аналогичные алгоритмы, рассмотрим лишь два наиболее типичных фрагмента. В первом случае на экран выводится толстая черная диагональная линия.

```
// рисование толстой черной диагональной линии
newpen.CreatePen(PS_SOLID,6,dwColor[0]);
oldpen-pDC->SelectObject(&newpen);
pDC->MoveTo(0,0);
pDC->LineTo(640,430);
pDC->TextOut(70,20,"<-diagonal 1ine",15);
// удаление пера
pDC->SelectObject(oldpen);
newpen.DeleteObjectO;
```

Перо создается функцией CreatePen(), которая задает рисование черных сплошных линий толщиной в шесть логических единиц. Сразу после этого функция SelectObject() обновляет перо, связывает его с контекстом устройства отображения (экраном монотора) и возвращает указатель на предыдущий объект пера. Функции LineTo() формируют диагональную линию, которая рисуется выбранными пером. Наконец, функция TextOut() выводит рядом с нарисованной фигурой надпись.

Работа с кистями организована аналогичным образом. В следующем коде создается кисть с заливкой в виде горизонтальных и вертикальных штрихов (HS_CROSS) синего цвета:

```
// рисование черного закругленного прямоугольника
// и заливка его синим цветом
newbrush.CreateHatchBrush(HS_CROSS,dwColor[3]);
 oldbrush=pDC->SelectObject(&newbrush);
 pDC->RoundRect(60,310,110,350,20,20);
 pDC->TextOut (120,310,"<-rounded rectangle",19);
 // удаление кисти
 pDC->SelectObject(oldbrush);
 newbrush.DeleteObject C);
```

Функция RoundRect () рисует прямоугольник с закругленными краями и заданными координатами, после чего выводится надпись. Все остальные фигуры рисуются аналогичным образом.

Задание 1.

1. Запустить Microsoft Visual Studio C++ .NET 2003/
2. Создать MFC проект под именем Gdi.
3. Набрать текст программы рисования фигур в файл CGdiView.cpp.
4. Создать файл Gdi.exe. Для этого необходимо выделить файл CGdiView.cpp. В меню Build выбрать Build Solution. Исправить ошибки компиляции.
5. Запустить программу, результаты работы программы проанализировать на экране дисплея.

Контрольные вопросы:

1. Какие принципы положены в основу библиотеки MFC?
2. Особенности MFC библиотеки?
3. Основной класс библиотеки MFC?
4. Какие файлы составляют проект в MFC приложении?
5. Как формируются цвета в программе рисования фигур?
6. Какие основные инструменты используются при рисовании фигур?

Лабораторная работа №26

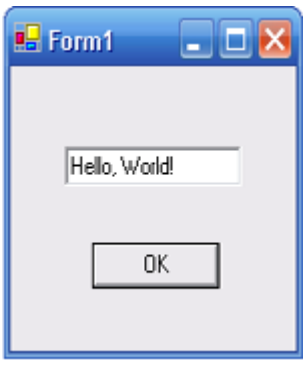
Тема: Создание приложения Windows.

Цель работы: *ознакомление со структурой проекта на Visual Basic, приобретение навыков программирования приложений windows на Visual Basic.*

Ход работы:

1. Выполнить задание в соответствии с указаниями
2. Ответить на контрольные вопросы
3. Предъявить преподавателю результаты работы: проект и исходный код
4. Оформить отчет в соответствии с ходом работы

Задание 1: создать проект на языке программирования Visual Basic.NET под названием "Hello, World" ("Здравствуй, мир!").

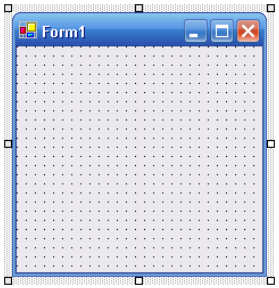


Этот проект после щелчка на кнопке OK выводит в текстовое поле сообщение: "Hello, World".

Создание нового проекта

1. Запустите Visual Studio.
2. На начальной странице Visual Studio на вкладке Проекты нажмите кнопку Создать проект. Новый проект можно начать и в меню Файл, выбрав в нем Создать - Проект. Появиться диалоговое окно Создать проект.
3. В этом окне нужно указать тип проекта, который необходимо создать, шаблон, который будет использоваться, имя проекта и расположение файлов.

- В списке Типы проектов выберите папку Проекты VisualBasic.
- В окне Шаблоны выберите шаблон Приложение для Windows. Visual Studio настроит среду разработки для программирования Windows-приложения на Visual Basic.NET.
- В текстовом поле Имя введите Hello_World в качестве имени проекта (в имени проекта лучше не использовать пробелы)
- В поле Расположение укажите путь для хранения файлов проекта. Выберите свою личную папку. В ней будет создана



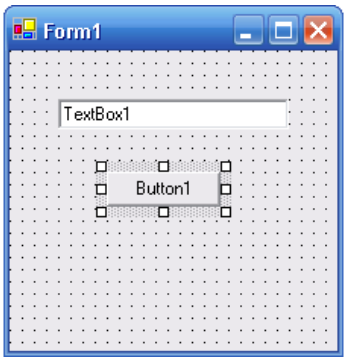
новая папка Hello_World, и проект "Hello_World" будет сохранен в созданной папке.

- Чтобы выбрать папку на диске, можно нажать кнопку Обзор.
- Нажмите ОК. Откроется новый созданный проект и на экране будет отображена пустая графическая форма Windows (обычно имеющая название Form1), на основе которой можно создавать интерфейс для пользователя вашего приложения.

4. Теперь мы увеличим форму, а затем создадим на ней текстовое поле и кнопку.

Создание пользовательского интерфейса

1. В Области элементов нажмите на элемент управления TextBox (текстовое поле), который находиться на закладке Windows Forms.
2. Нарисуйте текстовое поле. В этой программе текстовое поле используется для показа сообщения "Hello, world!" по нажатию на кнопку. Давайте добавим кнопку.
3. В Области элементов выберите элемент управления Button (кнопка).
4. На форме ниже текстового поля нарисуйте кнопку. Форма должна выглядеть примерно так.



Кнопка используется, когда пользователю нужно получить простейший отклик. Нажимая на кнопку, пользователь просит программу немедленно выполнить некоторое действие.

В терминах Visual Basic пользователь, нажимая на кнопку, создает событие, которое требуется обработать в программе.

Характеристики любой кнопки (как и всех объектов) можно изменить с помощью настройки свойств или обращения к её объекту в коде программы. В этом проекте мы будем настраивать свойства кнопки с помощью окна Свойства.

Настройка свойств

В Visual Basic все объекты, значит текстовое поле и кнопка тоже объекты. Текст в текстовом поле и надпись на кнопке - это свойства этих объектов. Для просмотра свойства объекта и их редактирования используется окно Свойства.

Окно Свойства находится в правой части экрана и отображает свойства выделенного объекта. По умолчанию - это сама форма. Чтобы отобразить свойства того объекта, который вы хотите редактировать, нужно его выделить щелчком.

Совет. Если окно свойств скрыто, вберите в меню Вид команду Окно свойств или нажмите клавишу F4.

Итак, вернемся к нашей задаче.

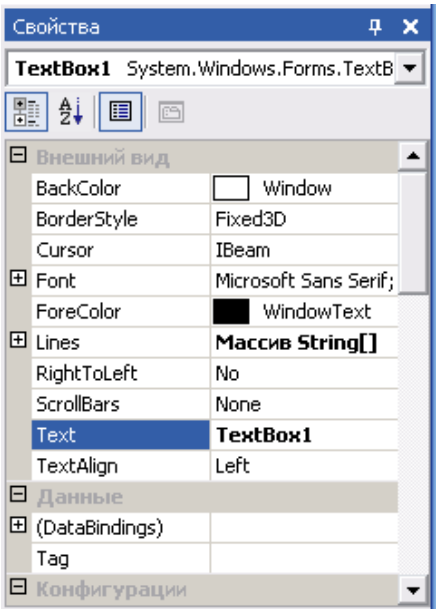
1. В окне Свойства задайте для текстового поля и кнопки приведенные ниже свойства. Значение "пусто" для TextBox1 означает, что вы должны удалить имеющееся значение и оставить свойство пустым. Установки, которые нужно ввести, показаны в кавычках. Сами кавычки вводить не нужно.

Объект	Свойство	Значение
TextBox1	Text	(пусто)
Button1	Text	"OK"

2. Чтобы изменить свойства объекта, выбираем объект на форме, в окне Свойства находим нужное свойство и вводим в его поле ввода нужное значение. После

этого нажимаем Enter или просто переходим на другую строку с другим свойством.

3. Начнем с изменения свойств текстового поля TextBox1.
   - Щелкните на текстовом поле TextBox1. Текстовое поле будет выделена и окружено маркерами изменения размера. В окне Свойства будут отображены свойства текстового поля.



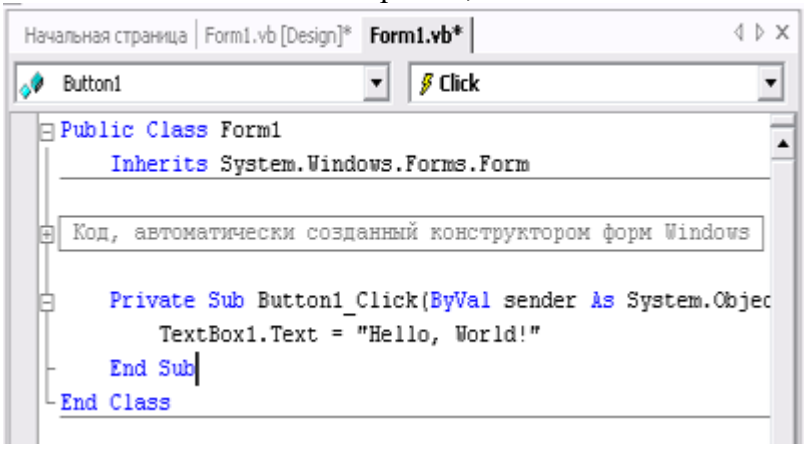
   - Так как свойств много, Visual Studio организует их по категориям и показывает в виде структуры. Если вы хотите увидеть свойства из какой-то категории, нажмите знак "плюс" (+) рядом с названием категории.
   - Прокрутите список окна Свойства до тех пор, пока не увидите свойство Text, расположенное в категории Внешний вид.
   - Дважды щелкните мышью в левом столбце свойства Text. Текущее значение свойства Text ("TextBox1") будет выделено, отредактируйте его с помощью клавиатуры.
4. Выполнив указанные действия, настроите свойства для второго объекта на форме - кнопки Button1.

Создание программного кода проекта

Для ввода и редактирования операторов в программе на Visual Basic используется Редактор кода. Редактор кода открывается двойным щелчком на объекте.

По условию задачи нам нужно, чтобы после щелчка на кнопке ОК в текстовое поле вывелось сообщение "Hello, World". Для этого напишем программный код в теле процедуры Button1Click() (обработчике события щелчок на кнопке).

1. В поле формы дважды щелкните на кнопке OK. В центральном окне среды разработки VisualStudio появиться Редактор кода, как показано ниже.



2. Когда кнопка Ok (объект "Button1") была помещена на поле формы, VisualStudio.Net автоматически добавил к коду этой формы процедуру Button1_Click для обработки основного для кнопки события - щелчок на кнопке. Процедура пустая, содержит только заголовок и оператор конца процедуры. В редакторе кода видны и другие

строки программы, которые среда VisualStudio добавила автоматически. Эти операторы находятся под общим заголовком "Код, автоматически созданный конструктором форм Windows".

3. Итак, тело процедуры должно находиться между строками:

Private Sub Button1_Click(ByVal sender As System.Object, ByVale As System.EventArgs) Handles Button2.Click

<...>

EndSub

4. Наберите перед строкой EndSub следующий программный код:

textBox1.Text = "Hello, World!"

Нажмите клавишу со стрелкой вниз, код, связанный с кнопкой Ok написан.

Совет. После того, как вы наберете имя объекта TextBox1 и точку, Visual Studio покажет список всех свойств объекта текстовое поле. Вы можете выбрать свойство из этого списка, дважды щелкнув на нем мышью, или набрать самостоятельно.

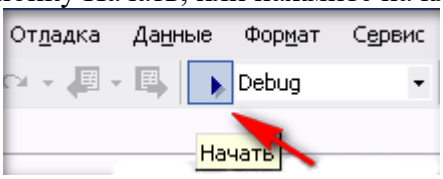
5. Во время работы программы оператор, который вы ввели, по нажатию на кнопку изменит свойство Text текстового поля на "Hello, world!".

Введенный выше оператор находиться внутри процедуры события - то есть, это инструкция, которая выполняется при щелчке на объекте Button1.

Теперь программу Hello World можно запускать.

Запуск программы Hello World.

1. Выберите из меню Отладка команду Начать (или на стандартной панели инструментов нажмите кнопку Начать, или нажмите на клавиатуре клавишу <F5>).



2. Программа будет скомпилирована и запустится в среде разработки Visual Studio.
3. Нажмите кнопку ОК. Программа покажет в тестовом поле приветствие "Hello, world!".Когда вы нажали ОК, код программы изменил свойство Text пустого текстового поля TextBox1 на текст 'Hello, world!" и показал этот текст в поле.
4. Чтобы сохранить изменения, на стандартной панели инструментов нажмите кнопку Сохранить все.

Задание 2. Написать программу для нахождения большего из двух чисел. Числа вводятся в текстовые поля. Ответ выводится в виде сообщения.

Пример: Упорядочить 3 числа по возрастанию. Числа вводятся в текстовые поля. Ответ выводится в виде сообщения.

Нарисуем на форме Form1 три элемента TextBox и одну кнопку CommandButton. По двойному щелчку на кнопке откроется редактор программного модуля с текстом

```
Private Sub Command1_Click()
End Sub
```

Полный текст модуля может быть таким:

```
Private Sub Exchange(ByRef x As Double, ByRef y As Double)
Dim temp As Double
temp = x
x = y
y = temp
End Sub

Private Sub Command1_Click()
Dim a As Double, b As Double, c As Double
a = Text1.Text
b = Text2.Text
c = Text3.Text
If a > b Then Exchange a, b
If c < b Then Exchange c, b
```

```
If a > b Then Exchange a, b
Text1.Text = a
Text2.Text = b
Text3.Text = c
End Sub
```

Контрольные вопросы:

1. Типы проектов в Visual Studio?
2. Виды шаблонов в Visual Studio?
3. Перечислите элементы управления, которые использовались при создании проета?
4. Как настроить свойства кнопки?
5. Как вызвать обработчик события для кнопки?
6. Где находится тело процедуры?
7. Как запустить программу?
8. Как вывести сообщение в Visual Basic?

Лабораторная работа №27

Тема: Примеры использования классов при создании приложений Windows.

Цель работы: *приобретение навыков использования классов при создании приложений Windows.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Каждое окно в Windows-приложении характеризуется определенными атрибутами, называемыми *классом окна*. В традиционной программе класс окна должен быть определен и зарегистрирован прежде, чем будет создано окно. При регистрации необходимо сообщить Windows, какой вид должно иметь окно и какую функцию оно выполняет. В то же время регистрация класса окна еще не означает создание самого окна. Для этого требуется выполнить дополнительные действия.

Структура Windows-программ отличается от структуры программ других типов. Это вызвано двумя обстоятельствами: во-первых, способом взаимодействия между программой и Windows, описанным выше; во-вторых, правилами, которым следует подчиняться для создания стандартного интерфейса Windows-приложения (т.е. чтобы сделать программу "похожей " на Windows-приложение).

Цель Windows - дать человеку, который хотя бы немного знаком с системой, возможность сесть за компьютер и запустить любое приложение без предварительной подготовки. Для этого Windows предоставляет дружественный интерфейс пользователя. Теоретически, если пользователь сумел запустить одно Windows-приложение, то он сумеет запустить и любое другое.

Одна из причин, по которой используют классы, является так называемое "повторное использование кода". Напомним, что в любом классе имеется хотя бы одна функция. Такие функции называются также "методами" или "членами" класса. Используя в своей работе тот или иной класс, мы делаем это потому, что нам нужны те или иные функции-члены этого класса.

Пример 1. Предположим, имеется некий класс, назовем его MyFirstClass, в котором содержатся две функции, рассчитывающие площади фигур RectangleArea и Cir-cleArea. Посмотрим, как эти функции можно вызывать из Windows-приложений.

1. Создаем новый проект, добавляем к нему модуль класса со следующим кодом:

```
Public Class MyFirstClass
Public a, b, r As Double

Public Function RectangleArea() As Double
Dim s As Double
s = a * b
Return s
End Function

Public Function CircleArea() As Double
Dim s As Double
s = Math.PI * Math.Pow(r, 2)
Return s
End Function


End Class
```

2. Далее, переносим на форму кнопку, дважды щелкаем по ней мышкой и в появившемся коде обработчика событий вводим следующий текст:

```
Dim obj As New MyFirstClass
Dim a, b, r, RectangleArea, CircleArea As Double
obj.a = 3
obj.b = 4
obj.r = 5
RectangleArea = obj.RectangleArea
CircleArea = obj.CircleArea
MsgBox("Rectangle Area is ... " & RectangleArea & " , Circle Area is ..." & CircleArea))
```

Как известно, обычно у функций бывают аргументы. В предыдущем примере было показано, как можно подставлять фактические значения аргументов функций. В первом случае переменные a, b и r имели глобальную область видимости (Public a As Double) и мы их инициализировали в вызывающем приложении (obj.a = 3).

Зачастую бывает желательно перед подстановкой аргумента убедиться в том, что он удовлетворяет определенным условиям (например, возраст человека не может быть отрицательным числом).

В таких случаях между вызывающим приложением и функциями-членами класса вводят особых посредников - так называемые свойства класса (Properties).

Свойства обычно являются общедоступными переменными класса, то есть они видны в вызывающем приложении, наряду с ними в классах обычно еще используются переменные с локальной областью видимостью (private), доступные только в данном классе, они являются аргументами функций-членов класса. Свойства состоят из двух разделов: Set и Get. Работа со свойствами производится следующим образом. В вызывающем приложении свойству присваивается некоторая величина, далее, в разделе Set свойства производится проверка области допустимых значений. Если проверка прошла хорошо, то это значение присваивается некой локальной переменной класса в разделе Get, которая, как мы знаем, является аргументом некой функции класса. В противном случае, или выдается сообщение об ошибке, или срабатывает так называемый обработчик исключений (об этом ниже). То есть, свойства класса являются некими шлюзами между вызывающим приложением и функциями-членами класса.

Пример 2. В следующем примере программа выдает на экран одно из семи изречений. Пользователь должен ввести любую цифру от 1 до 7.

Вернемся к нашему Windows-приложению. Добавим еще один класс (OracleClass) со следующим содержимым:

```
Public Class OracleClass
Private n As Integer
Property RandomNumber() As Integer
Get
RandomNumber = n
End Get
Set(ByVal Value As Integer)
If Value > 7 Then
MsgBox("Не выполняется область допустимых значений!!!")
Else
n = Value
End If
End Set
End Property
 Public Function Maxim(ByVal n As Integer) As String
 Dim Saying() As String = _
 {"Без труда не выловишь и рыбки из пруда", _
 "Ранней пташке Бог подает", _
 "Ум хорошо, а два лучше", _
 "Рука руку моет", _
 "Старый друг лучше новых двух", _
 "За одного битого двух небитых дают", _
 "Сам погибай, а товарища выручай"}
 Select Case n
 Case 1
 Return Saying(0)
 Case 2
 Return Saying(1)
 Case 3
 Return Saying(2)
 Case 4
 Return Saying(3)
 Case 5
 Return Saying(4)
 Case 6
 Return Saying(5)
 Case Else
 Return Saying(6)
 End Select

 End Function
 End Class
```

Затем, в форме размещаем поле вода со свойством Name txtNumber и кнопку, в обработчике событий которой вводим следующее:

```
Dim n As Integer
n = txtNumber.Text
Dim obj As New OracleClass()
obj.RandomNumber = n
Dim Adage As String
Adage = obj.Maxim(n)
MsgBox(Adage)
```

Общедоступная функция Maxim возвращает поговорку в строковом формате в зависимости от введенного пользователем номера. Если этот номер больше семи, то в разделе Set номер приравнивается нулю, выдается сообщение об ошибке, а функция возвращает первую фразу.

Задание 1:

1. Разработайте программу, находящую площадь трапеции $S = \frac{a+b}{2} \cdot h$.
2. Разработайте программу, находящую площадь ромба $S = \frac{1}{2} d_1 \cdot d_2$.
3. Разработайте программу, находящую площадь параллелограмма S=a*h.
4. Разработайте программу, находящую площадь треугольника по трем сторонам $S = \sqrt{p(p-a)(p-b)(p-c)}$, где $p = \frac{a+b+c}{2}$.

Задание 2:

1. Разработайте программу, выводящую на экран имена семи гномов из сказки "Белоснежка". (У гномов были имена соответствующие их характерам: Док, Тихоня, Соня, Чихун, Весельчак, Простак, Ворчун.)
2. Разработайте программу, выводящую на экран 6 материков.
3. Разработайте программу, выводящую на экран 7 чудес света. (Египетские пирамиды в Эль-Гизе, Висячие сады Семирамиды, Зевса Олимпийского статуя, Мавзолей в Галикарнасе, Артемиды Эфесской храм, Фаросский маяк, Колосс Родосский.)

Контрольные вопросы:

1. Для чего используется класс окна?
2. Каким оператором создается функция?
3. Каким оператором создается класс?
4. Каким оператором задаются свойства класса?
5. Как задаются переменные, доступные только в данном классе?
6. Из каких разделов состоят свойства?

Лабораторная работа №28

Тема: Приложения, обрабатывающие клавиатурные сообщения, сообщения от драйвера "мыши" и таймера.

Цель работы: *приобретение навыков создания приложений, обрабатывающих сообщения от таймера.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Для создания событий удобно использовать диалог Add Procedure в меню Tools.

Пример 1: программа, которая будет проверять нажатия некоторых клавиш. На форму ничего не размещать, вот код:

Private Sub Form1_KeyDown(ByVal sender As Object, ByVal e As System.Windows.Forms.KeyEventArgs) Handles MyBase.KeyDown

If e.Control = True And e.KeyCode = Keys.Q Then MsgBox("Нажата Control и Q")

' Если нажата клавиша Contro(вместо неё можно написать Alt или Shift) и Q то выдаем сообщение

End Sub

Свойство KeyCode не умеет обрабатывать русские буквы.

Пример 2: Для обработки русских и прописных букв надо использовать событие KeyPress и свойство KeyChar. Вот код:

Private Sub Form1_KeyPress(ByVal sender As Object, ByVal e As System.Windows.Forms.KeyPressEventArgs) Handles MyBase.KeyPress

If e.KeyChar = "n" Then MsgBox("Нажата клавиша: n")

' Если нажата клавиша 'n' то выводим сообщение

End Sub

Пример 3: сделаем программу, которая будет узнавать координаты курсора. На форме размести 2 метки, и таймер(Interval=100, Enabled=True). Код:

Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Timer1.Tick

' В метки записываем координаты курсора по оси X и Y

Label1.Text = Cursor.Position.X

Label2.Text = Cursor.Position.Y

End Sub

Пример 4: Теперь сделаем программу, которая будет перемещать курсор. На форме разместить кнопку. Код программы:

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

' При нажатии на кнопку передвигаем курсор(Point переводится как пункт, New - новый)

Cursor.Position = New Point(100, 100)

End Sub

У каждого видимого объекта есть свойство Cursor. Оно нужно для того, чтобы при наведении курсора на объект, у него менялся курсор.

Пример 5: Сейчас сделаем программу, в которой при наведении на текстовое поле курсора, курсор менялся на стандартный курсор Hand. На форме разместите текстовое поле, код:

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

' При загрузке формы, у текст. поля меняем курсор на стандартный. Объект Cursors содержит все стандартные курсоры.

TextBox1.Cursor = Cursors.Hand

End Sub

Пример 6: пример того как можно скрыть курсор мыши.

ВАРИАНТ №1

СОЗДАЙТЕ МОДУЛЬ.

Private Declare Function ShowCursor Lib "user32" _

```
    (ByVal bShow As Long) As Long
     Const HIDE_CURSOR = (0)
    Const SHOW_CURSOR = (1)
  Public Function ShowCur()
       Call ShowCursor(SHOW_CURSOR)
  End Function
  Public Function HideCur()
    Call ShowCursor(HIDE_CURSOR)
  End Function
```

СОЗДАЙТЕ ФОРМУ

```
  Private Sub Command1_Click()
    ShowCur 'Показываем курсор
  End Sub
  Private Sub Command2_Click()
    HideCur 'Скрываем курсор
  End Sub
```

ВАРИАНТ №2

КОД ФОРМЫ

```
  Private Declare Function ShowCursor Lib "user32" _
          (ByVal bShow As Long) As Long
  Private Sub Command1_Click()
    Call ShowCursor(0) 'Скрываем курсор
  End Sub
  Private Sub Command2_Click()
    Call ShowCursor(1) 'Показываем курсор
  End Sub
```

Пример 7: В этом примере представлены коды основных клавиш. Например, у CTRL вообще не определяется ASCII код.

vbKeyF1 - От F1
...
vbKeyF12 - До F12
vbKeyA - От A
...
vbKeyZ - До Z(только английские буквы (заглавные и обычные))
vbKeyBack - BackSpace
vbKeyInsert - Insert
vbKeyHome - Home
vbKeyPageUp - Page Up
vbKeyDelete - Delete
VbKeyEnd - End
VbKeyPageDown - Page Down
vbKeyNumlock - Num Lock
vbKeyCapital - Caps Lock
vbKeyEscape - Esc
vbKeyReturn - Enter
vbKeySpace - Пробел
vbKeyShift - Shift
vbKeyTab - TAB
VbKeyControl - CTRL
vbKeyMenu - ALT
VbKeyLeft - Стрелка влево
VbKeyRight - Стрелка в право
VbKeyDown - Стрелка в низ
VbKeyUp - Стрелка вверх

Код:

```
Private Sub Text1_KeyDown(KeyCode As Integer, Shift As Integer)
If KeyCode = vbKeySpace Then MsgBox "Нажат пробел"
' Если нажат пробел то выскакивает сообщение
End Sub
```

Пример 8: Запрет ввода определенных символов.

На форму добавим текстовое поле, откроем код проекта, выберем текстовое поле и событие KeyPress. Это событие наступает, когда нажата клавиша.

Внутри события напишем этот текст:

```
Select Case KeyAscii
Case 0 To 192 - код символа
Case Else
KeyAscii = 0
End Select
```

0 To 192 - код символа латинских символов.

Case Else - в противном случае

KeyAscii = 0 - не вводить дальше.

Запрещаем ввод латинских символов:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
   Select Case KeyAscii
      Case 192 To 256
      Case Else
      KeyAscii = 0
   End Select
End Sub
```

Запретит всех символов кроме номера, 0-9 :

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
   Select Case KeyAscii
      Case 48 To 57
      Case Else
      KeyAscii = 0
   End Select
End Sub
```

Задание 1: Написать программу, которая будет проверять нажатия клавиш «S» и «Y».

Задание 2: Написать программу, которая обрабатывает нажатие русских и прописных букв «А» и «д».

Задание 3: Создать программу, которая будет узнавать координаты курсора, и выводить их в текстовые поля. Добавить на форму кнопку остановки таймера.

Задание 4: Создать программу, в которой при наведении на кнопку курсора, курсор менялся на стандартный курсор Hand.

Задание 5: Создать программу, которая выводит сообщение *«Нажата клавиша Shift»* при нажатии клавиши *Shift.*

Контрольные вопросы:

1. Какие события обработки клавиатурных сообщений использовались в созданных вами программах?
2. Какое свойство курсора отвечает за его положение?
3. Какое свойство курсора отвечает за его скрытие?
4. Как запретить ввод латинских символов?

Лабораторная работа №29

Тема: Использование ресурсов в приложениях Windows.

Цель работы: *приобретение навыков программирования приложений Windows с использованием ресурсов.*

Ход работы:

1. Изучить теоретическую часть
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

Элемент управления LinkLabel позволяет создавать в форме ссылки на Web-страницы, подобно гиперссылкам в HTML-документах, ссылки на открытие файлов какими-либо программами, ссылки на просмотр содержания логических дисков, папок и проч.

Пример 1. Напишем программу, которая с помощью элемента управления LinkLabel обеспечит ссылку для посещения почтового сервера www.mail.ru, ссылку для просмотра папки C:\Windows\ и ссылку для запуска текстового редактора Блокнот.

Для программирования этой задачи после запуска VB10 выберем шаблон Windows Form Application, затем из панели Toolbox перетащим на форму три элемента управления LinkLabel. Равномерно разместим их на форме. Все начальные значения свойств элементов LinkLabel укажем в программном коде при обработке события загрузки формы Form1_Load:

```
' Программа обеспечивает ссылку для посещения почтового сервера
' www.mail.ru, ссылку для просмотра папки C:\Windows\ и ссылку для
' запуска текстового редактора Блокнот с помощью элемента управления
' LinkLabel
Public Class Form1
  Private Sub Form1_Load(ByVal sender As System.Object,
                         ByVal e As System.EventArgs) Handles MyBase.Load

    Me.Text = "Щелкните по ссылке:"
    LinkLabel1.Text = "www.mail.ru"
    LinkLabel2.Text = "Папка C:\Windows\"
    LinkLabel3.Text = "Вызвать " & ChrW(34) & "Блокнот" & ChrW(34)
    Me.Font = New System.Drawing.Font("Courier New", 12.0!)
    LinkLabel1.LinkVisited = True
    LinkLabel2.LinkVisited = True
    LinkLabel3.LinkVisited = True
  End Sub

  ' Обработка события "щелчок на какой-либо ссылке"
  Private Sub Link_Clicked(ByVal sender As System.Object, ByVal e As  _
          System.Windows.Forms.LinkLabelLinkClickedEventArgs) _
               Handles LinkLabel1.LinkClicked, LinkLabel2.LinkClicked,
                       LinkLabel3.LinkClicked


   Dim s As String = sender.ToString.Substring(38, 1)
   ' – или ...Ctype(sender, Button).Text...
   Select Case s    ' Выбор ссылки:
     Case "w"
```

```
            System.Diagnostics.Process.Start(
                                    "IExplore.exe", "http://www.mail.ru")
        Case "П"
            System.Diagnostics.Process.Start("C:\Windows\")
        Case "В"
            System.Diagnostics.Process.Start("Notepad", "text.txt")
    End Select
  End Sub
End Class
```

Как видно из программного кода, в свойстве Text каждой из ссылки LinkLabel задаем текст, из которого пользователь поймет назначение каждой ссылки. В задании свойства Text ссылки LinkLabel3 для того, чтобы слово «Блокнот» было в двойных кавычках, используем функцию ChrW(34). Эта функция возвращает символ, соответствующий указанному коду (34) символа в кодировке Unicode. В данном случае функция ChrW(34) возвращает символ “двойная кавычка”. Для большей выразительности задаем шрифт Courier New, 12 пунктов. Поскольку свойство LinkVisited=True, то соответствующая ссылка отображается как уже посещенная (изменяется цвет).

Чтобы обрабатывать событие Click по каждой из ссылок, создадим одну процедуру. Назовем эту процедуру Link_Clicked(), а после ключевого слова Handles перечислим через запятую все три события, которые мы хотим обрабатывать в этой процедуре.

Далее в зависимости от объекта (ссылки), создающего события (LinkLabel1, LinkLabel2, LinkLabel3), мы вызываем одну из трех программ: либо Internet Explorer, либо Windows Explorer, либо Блокнот. Информация об объекте, создающем событие Click, записана в объектную переменную sender. Она позволяет распознавать объекты (ссылки), создающие события. Чтобы «вытащить» эту информацию из sender, в строковом представлении (ToString) объектной переменной sender с помощью Substring выделим подстроку с параметрами (38,1). Указанные параметры означают, что следует выделить в подстроку один символ, начиная с 38-й позиции.

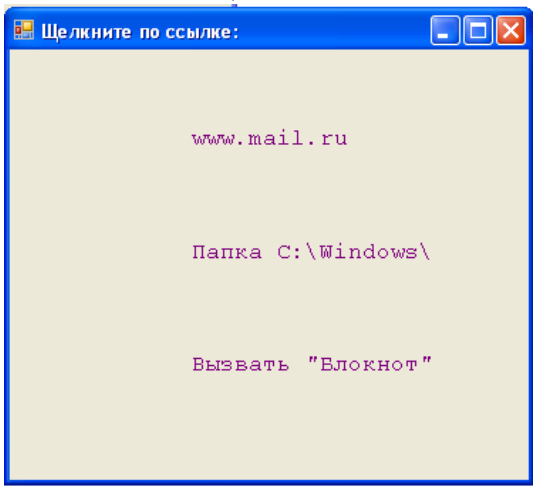


Рис. Ссылки на ресурс

На 38-м месте находится первая буква свойства Text каждой из ссылок, например для первой ссылки www.mail.ru, на 38-м месте находится буква «w». По первым буквам каждой из ссылок идентифицируем ссылку и с помощью метода Start вызываем либо Internet Explorer, либо Windows Explorer, либо Блокнот. Вторым параметром метода Start является имя ресурса, подлежащее открытию. Именем ресурса может быть или название Web-страницы, или имя текстового файла.

Вызов (запуск) исполняемых файлов помимо метода Start может осуществлен также командой Shell, например, таким образом:

```
Shell("notepad C:\text1.txt", AppWinStyle.MaximizedFocus)
```

Задание 1:

Напишем программу, которая с помощью элемента управления LinkLabel обеспечит ссылку для посещения почтового сервера www.ya.ru, ссылку для просмотра папки “Мои рисунки” и ссылку для запуска Калькулятора. Добавить на форму кнопку закрытия формы.

Контрольные вопросы:

1. Что представляют собой ресурсы в приложениях Windows?
2. С помощью какого элемента управления создают ссылку на нужный ресурс?
3. Для чего в программе использовали функцию ChrW(34)?
4. Какое свойство элемента LinkLabel отвечает за посещение ссылки?
5. Какую функцию выполняла процедура Link_Clicked()?

Лабораторная работа №30

Тема: Использование ресурсов в приложениях Windows.

Цель работы: *приобретение навыков программирования приложений Windows с использованием органов управления операционной системой.*

Ход работы:

1. Выполнить задание в соответствии с указаниями
2. Ответить на контрольные вопросы
3. Предъявить преподавателю результаты работы: проект и исходный код
4. Оформить отчет в соответствии с ходом работы

Задание 1. Как узнать имя компьютера и имя пользователя?

```
Private Declare Function GetComputerNameA Lib "kernel32" (ByVal lpBuffer As String, nSize As Long) As Long
Private Declare Function WNetGetUserA Lib "mpr.dll" (ByVal lpName As String, ByVal lpUserName As String, lpnLength As Long) As Long
Function GetComputerName() As String
Dim sBuffer As String * 255
If GetComputerNameA(sBuffer, 255&) <> 0 Then
GetComputerName = Left$(sBuffer, InStr(sBuffer, vbNullChar) - 1)
End If
End Function
Function GetUserName() As String
Dim sUserNameBuff As String * 255
sUserNameBuff = Space(255)
Call WNetGetUserA(vbNullString, sUserNameBuff, 255&)
GetUserName = Left$(sUserNameBuff, InStr(sUserNameBuff, vbNullChar) - 1)
End Function

Private Sub Command1_Click()
MsgBox GetComputerName, 64, "ComputerName"
MsgBox GetUserName, 64, "GetUserName"
End Sub
```

Задание 2. Как изменить имя компьютера?

```
'добавьте модуль
Declare Function SetComputerName Lib "kernel32" Alias "SetComputerNameA" (ByVal lpComputerName As String) As Long
'добавьте кнопку
a$ = "Hello World"
b& = SetComputerName(a$)
```

Задание 3. Программно переключить клавиатуру с русского на английский и обратно - Visual Basic

Расположите на форме элемент CommandButton

```
Private Declare Function ActivateKeyboardLayout Lib "user32" (ByVal HKL As Long, ByVal flags As Long) As Long
Private Sub Command1_Click()
ActivateKeyboardLayout 0, 0
End Sub.
```

Задание 4. Как узнать, сколько работает ваш компьютер?

```
Private Declare Function GetTickCount Lib "kernel32" () As Long
Private Sub Form_Load()
Dim a_hour, a_minute, a_second
a = Format(GetTickCount() / 1000, "0") 'всего секунд
a_hour = Int(a / 3600)
a = a - a_hour * 3600
a_minute = Int(a / 60)
a_second = a - a_minute * 60
MsgBox "Ваш компьютер работает в эту загрузку " & Str(a_hour) & " часов " & Str(a_minute) & " минут" & Str(a_second) & " секунд"
End Sub
```

Задание 5. Очистить/показать содержимое корзины.

Данный код вызывает окно "Очистить содержимое корзины?"

Расположите на форме 2 элемента CommandButton.

```
Private Declare Function SHEmptyRecycleBin Lib "shell32.dll" Alias "SHEmptyRecycleBinA" (ByVal hWnd As Long, ByVal pszRootPath As String, ByVal dwFlags As Long) As Long
Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal hWnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long
Const SHERB_NOPROGRESSUI = &H2
Const SW_SHOWNORMAL As Long = 1
Private Sub Command1_Click()
Call SHEmptyRecycleBin(Me.hWnd, "", SHERB_NOPROGRESSUI)
End Sub
Private Sub Command2_Click()
Dim success As Long
success = ShellExecute(h, "Open", "explorer.exe", "/root,::{645FF040-5081-101B-9F08-00AA002F954E}", 0&, SW_SHOWNORMAL)
End Sub
```

Задание 6. Установить дату и время на компьютере.

```
Private Type SystemTime
wYear As Integer
wMonth As Integer
wDayOfWeek As Integer
wDay As Integer
wHour As Integer
wMinute As Integer
wSecond As Integer
wMilliseconds As Integer
End Type
Private Declare Function SetLocalTime Lib "kernel32.dll" (lpSystemTime As SystemTime) As Long

Public Sub SetNewTime(NewHour As Integer, NewMinute As Integer, NewSecond As Integer)
Dim SetTime As SystemTime
```

```
Dim RetVal As Long
SetTime.wHour = NewHour
SetTime.wMinute = NewMinute
SetTime.wSecond = NewSecond
SetTime.wMilliseconds = 0
SetTime.wDay = Day(Date)
'SetTime.wDay = 1
SetTime.wMonth = Month(Date)
'SetTime.wMonth = 1
SetTime.wYear = Year(Date)
'SetTime.wYear = 1990
RetVal = SetLocalTime(SetTime)
End Sub

Private Sub Command1_Click()
Call SetNewTime(13, 20, 50)
End Sub
```

4. Информационное обеспечение обучения

Перечень рекомендуемых учебных изданий, Интернет-ресурсов, дополнительной литературы

Основные источники:

1. Федорова Г.Н. Разработка программных модулей программного обеспечения для компьютерных систем / Г.Н. Федорова. – М.: Издательский центр «Академия», 2016. – 286 с.
2. Тюгашев А.А. Языки программирования. Учебное пособие. Стандарт третьего поколения / А.А. Тюгашев – М.: Издательский дом «Питер», 2017. – 562 с.
3. Орлов С.А. Теория и практика языков программирования / С.А. Орлов. – М.: Издательский дом «Питер», 2014. – 688 с.
4. Паттерсон Д., Хеннеси Д. Программирование. Архитектура компьютера и проектирование компьютерных систем. 4-е издание / Д. Паттерсон – М.: Издательский дом «Питер», 2012. – 508 с.

Дополнительные источники:

1. Абель П. Язык Ассемблера для IBM PC и программирования / П. Абель. – М.: М.: Высш. шк., 1992. – 650 с.
2. Лав Р. Системное программирование / Р. Лав. – М.: Издательский дом «Питер», 2017. – 662 с.
3. Галисеев Г.В. Ассемблер IBM PC. Самоучитель / Г.В. Галисеев – М: Вильямс, 2008. – 438 с.
4. Юров В.И. Справочник по языку Ассемблера IBM PC / В.И. Юров – М.: Издательский дом «Питер», 2004. – 422 с.
5. Захаров Д.В. Системное программирование / Д.В. Захаров – М.: Издательство НТЛ, 2007. – 502 с.
6. Пирогов В.Ю. Assembler. Учебный курс / В.Ю. Пирогов – М.: М.: Нолидж, 2008. – 680 с.
7. Побегайло А.П. Системное программирование в Windows / А.П. Побегайло– М.: Издательство НТЛ, 2007. – 402 с.

Интернет- ресурсы:

1. http://nick-yk.narod.ru/doc/system.htm.
2. http://eugene.eto-ya.com/ucheba/sistemnoe-programmirovanie-polnyj-kurs/
3. http://www.uhlib.ru/kompyutery_i_internet/sistemnoe_programmirovanie_v_srede_windows/index.php