МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

НЕВИННОМЫССКИЙ ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)

Методические указания по выполнению лабораторных работ

по дисциплине

«Управляющие микропроцессорные комплексы»

Методические указания к выполнению лабораторных работ
Направление подготовки 15.04.04
«Автоматизация технологических процессов и производств»
Направленность (профиль) «Информационно-управляющие системы»

Невинномысск 2024

Методические указания предназначены для студентов очно-заочной формы обучения специальности 15.04.04 Автоматизация технологических процессов и производств и других технических специальностей. Они содержат основы теории, порядок проведения лабораторных работ и обработки экспериментальных данных, перечень контрольных вопросов для самоподготовки и список рекомендуемой литературы. Работы подобраны и расположены в соответствии с методикой изучения дисциплины «Управляющие микропроцессорные комплексы». Объем и последовательность выполнения работ определяются преподавателем в зависимости от количества часов, предусмотренных учебным планом дисциплины, как для очной, так и для заочной форм обучения.

Методические указания разработаны в соответствии с требованиями Федерального Государственного образовательного стандарта в части содержания и уровня подготовки выпускников по специальности Управляющие микропроцессорные комплексы

Код	Формулировка
ПК-3	Способность: составлять описание принципов действия и конструкции устройств, проектируемых технических средств и систем автоматизации, управления, контроля, диагностики и испытаний технологических процессов и производств общепромышленного и специального назначения для различных отраслей национального хозяйства; проектировать их архитектурно-программные комплексы

Составитель: канд. техн. наук, доцент Ю.Н. Кочеров

Ответственный редактор: канд. техн. наук, доцент Д.В. Болдырев

Содержание

Лабораторная работа № 1 4
Лабораторная работа № 2 18
Лабораторная работа № 3 28
Лабораторная работа № 4 37
Лабораторная работа № 5 48
Лабораторная работа № 6 56
Лабораторная работа № 7 61
Рекомендуемая литература 69
Приложение А 70
Приложение Б 76
Приложение В 80

Лабораторная работа № 1

Создание простого оконного приложения

Цель работы: Получить навык создания оконных приложений на языке ассемблера 32-хразрядного процессора Intel.

Краткие сведения из теории

Минимально оконное приложение Windows состоит из трех частей.

1. Выполнение стартового кода.

2. Выполнение главной функции, которая выполняет следующие действия:

2.1. регистрирует класс окна;

2.2. создает окно;

2.3. отображает окно;

2.4. запускает цикл обработки сообщений;

2.5. завершает выполнение приложения.

3. Организация обработки сообщений в оконной процедуре.

Выполнение любого оконного Windows-приложения начинается с главной функции. Она содержит код, осуществляющий настройку (инициализацию) приложения в среде операционной системы Windows. Видимым для пользователя результатом работы главной функции является появление на экране графического объекта в виде окна. Последним действием кода главной функции является создание цикла обработки сообщений. После его создания приложением становится пассивным и начинает взаимодействовать с внешним миром посредством специальным образом оформленных данных – сообщений. Обработка поступающих приложению сообщений осуществляется специальной процедурой, называемой оконной. Оконная процедура уникальна тем, что может быть вызвана только из операционной системы, а не из приложения, которое ее содер-

жит (функция обратного вызова). Тело оконной процедуры также имеет определенную структуру.

Вызов функций Win32 API осуществляется аналогично вызову внешних функций, а передача параметров осуществляется через стек. Передача параметров производится в соответствии со стилем stdcall – параметры в стек помещается справа налево, т.е. первым в стек идет последний параметр функции, и удаляются из стека по завершению работы процедуры.

Модель сегментации, используемая в программах Win32 – flat, плоская модель памяти. В соответствии с этой моделью компилятор создает программу, которая содержит одни 32-разрядный сегмент для данных и кода программы. Код загрузочного модуля будет работать на процессорах i386 и выше. По этой причине директиве .MODEL должна предшествовать одна из директив: .386, .486 или .586.

В программе с плоской моделью памяти используется адресация программного кода и данных типа near. Это же касается и атрибута расстояния в директиве PROC, которые также имеет тип near.

Функции Win32 API должны быть объявлены внешними с помощью директивы EXTERN. Например,

```
EXTERN   MessageBoxA@16: NEAR
EXTERN   CreateWindowExA@48: NEAR
EXTERN   DefWindowProcA@16: NEAR
```

Рассмотрим запись имени функции, например функции вывода на экран сообщения с кнопкой MessageBox: MessageBoxA@16.

Для работы с однобайтной (ANSI) и двухбайтной (UNICODE) кодировкой символов программный интерфейс Win32 API имеет два варианта функций, которые различаются последним символом в названии. Ес-

ли это «A», то данная функция работает в кодировке ANSI, если «W» – UNICODE.

Наличие суффикса «Ex», говорит об использовании расширенного варианта функций, которые обладают дополнительными возможностями по сравнению со старыми вариантами функций Win32 API. Исчерпывающим источником информации по функциям Win32 API является MSDN (Microsoft Developer Network – информационная система поддержки разработчика по продуктам Microsoft), его можно найти в Интернете по адресу http://www.microsoft.com/msdn/.

При использовании MASM необходимо также в конце имени функции добавить @N, где N – размер всех параметров в байтах, передаваемых функции при ее вызове. Размер одного параметра – четыре байта.

Стартовый код

Стартовый код представляет собой последовательный вызов функций Win32 API, которые могут быть использованы для анализа информации о версии Windows, получения указателя на командную строку, получения указателя на блок с переменными окружения, получения значения базового адреса (дескриптора), по которому загружен модуль и т.д.

Функция GetModuleHandle (ModuleName: PChar): THandle считывает описатель модуля. Ее единственный параметр – адрес ASCIIZ-строки с именем исполняемого файла, базовый адрес загрузки которого необходимо получить.

Поскольку в адресное пространство процесса можно загрузить несколько файлов, для работы с ними требуется механизм их однозначной идентификации. При загрузке исполняемого файла в адресное пространство процесса ему присваивается уникальный номер. Этот номер (Handle Instance, hInst – описатель экземпляра) используется при вызове многих

функций Win32 API, загружающих те или иные ресурсы для данной программы. Значение hInst равно базовому адресу в адресном пространстве процесса, по которому загружен данный файл с расширением .exe.

При вызове функции GetModuleHandleA@4, передав ей значение NULL, получим значение hInst для текущей программы.

Регистрация класса окна

Под классом окна понимается совокупность присущих ему характеристик, таких как стиль его границ, формы указателя мыши, значков, цвет фона, наличие меню, адрес оконной процедуры, обрабатывающей сообщения этого окна.

Регистрация класса окон осуществляется с помощью функции RegisterClassA, единственным параметром которой является указатель на структуру WNDCLASS, содержащую информацию об окне.

```
WNDCLASS STRUC
    CLSSTYLE        DD  ?   ;стиль окна
    CLWNDPROC       DD  ?   ;указатель на процедуру окна
    CLSCSEXTRA      DD  ?   ;информация о доп. байтах для
                            ;данной структуры
    CLWNDEXTRA      DD  ?   ;информация о доп. байтах для окна
    CLSHINSTANCE    DD  ?   ;дескриптор приложения
    CLSHICON        DD  ?   ;идентификатор иконы окна
    CLSHCURSOR      DD  ?   ;идентификатор курсора окна
    CLBKGROUND      DD  ?   ;идентификатор кисти окна
    CLMENUNAME      DD  ?   ;имя-идентификатор меню
    CLNAME          DD  ?   ;специфицирует имя класса окон
WNDCLASS ENDS
```

Таблица 1.1 – Константы стиля окна

Константа	Значение
CS_BYTEALIGNCLIENT	Использование границы по байту по оси X. Выравнивание клиентской области окна
CS_BYTEALIGNWINDOW	Использование границы по байту по оси X. Выравнивание окна
CS_CLASSDC	Классу окна присваивается собственный контекст изображения, который можно разделить между копиями.
CS_DBCLKS	Окну будут посылаться сообщения о двойном щелчке кнопки «мыши».
CS_GLOBALCLASS	Определяется глобальный класс окон.
CS_HREDRAW	Обеспечивается перерисовка содержимого клиентской области окна при изменении размера окна по горизонтали.
CS_KEYCVTWINDOW	Будет выполняться преобразование виртуальных клавиш.
CS_NOCLOSE	В системном меню блокируется выбор пункта для закрытия окна.
CS_NOKEYCVT	Отключается преобразование виртуальных клавиш.
CS_OWNDC	Каждому экземпляру окна присваивается собственный контекст изображения.
CS_PARENTDC	Классу окна передается контекст изображения родительского окна.
CS_SAVEBITS	Часть изображения на экране, закрытая окном, сохраняется.
CS_VREDRAW	Обеспечивается перерисовка содержимого клиентской области окна при изменении размера окна по вертикали.

Поле CLSSTYLE определяет стиль границ окна и его поведение при перерисовке. Значения стиля является целочисленным и формируется из констант (таблица 1.1). Константы данного поля и других полей можно найти в файле C:\masm32\INCLUDE\RESOURCE.H.

Поле CLWNDPROC – значение указателя на функцию окна, которая выполняет все задачи, связанные с окном.

Поле CLSCSEXTRA и CLWNDEXTRA служат для указания количества байтов, дополнительно резервируемых в структуре класса окна WNDCLASS и выделяемых для всех дополнительных структур, которые создаются с использованием данного класса окна. Обычно эти поля инициализированы нулевыми значениями (NULL).

В поля CLSHICON и CLSHCURSOR загружаются дескрипторы значка и указателя мыши. После запуска приложения значок будет отображаться на панели задач Windows и в левом верхнем углу окна приложения, а указатель мыши появится в области окна. Значки и указатели мыши представляют собой ресурсы и находятся в отдельных файлах. Для загрузки поименованного ресурса курсора используется функция

LoadCursor(Instance: THandle; CursorName: PChar): HCursor,

параметрами которой являются Instance – экземпляр модуля, исполнимый файл которого содержит курсор, или 0 для предопределенного курсора; CursorName – строка (заканчивающаяся пустым символом) или имя целочисленного идентификатора или предопределенный курсор, определенный одной из констант idc_.

Для загрузки поименованного ресурса пиктограммы используется функция

LoadIcon(Instance: THandle; IconName: PChar): HIcon,

параметрами которой являются Instanc – экземпляр модуля, исполнимый файл которого содержит пиктограмму, или 0 для предопределенной пиктограммы; IconName – строка или имя целочисленного идентификатора или предопределенная пиктограмма, определенная одной из констант idi_.

Поле CLBKGROUND определяет кисть, используемую для закраски фона окна. Значением данного параметра может быть как идентифика-

тор физической кисти, так и значение цвета (от 1 до 29). При использовании значения цвета нужно выбрать одно из следующего списка и прибавить к нему 1 (значения данных констант можно найти в файле C:\masm32\INCLUDE\WINDOWS.INC):

COLOR_SCROLLBAR; COLOR_BACKGROUND;
COLOR_ACTIVECAPTION; COLOR_iNACTIVECAPTION;
COLOR_MENU; COLOR_WINDOW; COLOR_WINDOWFRAME;
COLOR_MENUTEXT; COLOR_WINDOWTEXT;
COLOR_CAPTIONTEXT; COLOR_ACTIVEBORDER;
COLOR_INACTIVEBORDER; COLOR_APPWORKSPACE;
COLOR_HIGHLIQHT; COLOR_HIGHLIGHTTEXT;
COLOR_BTNFACE; COLOR_BTNSHADOW;
COLOR_GRAYTEXT; COLOR_DTNTEXT;
COLOR_INACTIVECAPTIONTEXT; COLOR_BTNHIGHLIGHT.

В поле CLMENUNAME записывается указатель на ASCIIZ-строку с именем меню. Если меню не используется, то в поле записывается значение NULL.

Поле CLNAME содержит длинный указатель на ASCIIZ-строку, которая определяет имя класса. Имя класса должно быть уникальным, чтобы не возникало проблем при разделении класса между приложениями.

После инициализации структуры регистрируется класс окна в системе. После регистрации класса окна структура WNDCLASS больше не нужна.

Создание окна

На основе зарегистрированного класса с помощью функции CreateWindowExA (или CreateWindowA) можно создать экземпляр окна.

Функция CreateWindow(ExStyle: Longint; ClassName, WindowName: PChar; Style: Longint; X, Y, Width, Height: Integer; WndParent: HWnd; Menu: HMenu; Instance: THandle; Param: Pointer):HWnd
создает перекрытое, всплывающее или дочернее окно с расширенным стилем.

Параметры функции CreateWindowExA:

ExStyle – один из следующих расширенных стилей окна: ws_ex_DlgModalFrame, или ws_ex_NoParentNotify. Позволяет задать дополнительные стили окна;

ClassName – указатель на ASCIIZ-строку с именем класса окна или предопределенное имя класса органа управления;

WindowName – указатель на ASCIIZ-строку с текстом, помещаемым в заголовок окна;

Style – одна из констант стиля окна или органа управления или их комбинация. К этим константам относятся константы ds_, ws_, bs_, cbs_, es_, lbs_, sbs_, ss_;

X, Y: – начальное положение окна или cw_UseDefault (80000000h);

Width – Начальная ширина окна (в единицах устройства);

Height – начальная высота окна (в единицах устройства);

WndParent – дескриптор родительского окна. Между двумя окнами Windows-приложения можно установить родовые отношения. Дочернее окно всегда должно появляться в области родительского окна;

Menu – идентификатор главного меню или дочернего окна;

Instance – дескриптор приложения создающего окно;

Param – используется при создании окна для передачи данных или указателя на них в оконную функцию. Все параметры, передаваемые функцией CreateWindowExA, сохраняются в создаваемой Windows внут-

ренней структуре TCreateStruct. Поля этой структуру идентичны параметрам функции CreateWindowExA. Указатель на структуру TCreateStruct передается оконной функции при обработке сообщения wm_Create. Сам указатель находится в поле lParam сообщения. Значение параметра Param функции CreateWindowExA находится в поле lCreateParams структуры TCreateStruct. Для создания дочернего окна MDI должно быть указателем на структуру TClientCreateStruct.

В случае успешного завершения функция возвращает дескриптор окна, в противном случае – 0.

Отображение окна

Для появления созданного окна на экране необходимо применить функцию ShowWindowA:

ShowWindow(Wnd: HWnd; CmdShow: Integer),

которая отображает или прячет окно образом, указанным параметром CmdShow. Wnd – идентификатор окна. CmdShow – одна из констант sw_, в зависимости от значения которой окно отображается в стандартном виде, развернутом на весь экран, свернутым в значок и т.д.

Функция UpdateWindow(Wnd: HWnd) посылает сообщение wm_Paint прямо оконной функции данного окна, если область обновления окна не пуста.

Цикл обработки сообщений

Windows поддерживает очередь сообщений для каждого приложения. Запуск приложения автоматически подразумевает формирование для него очереди.

Формат всех сообщений Windows одинаков и описывается структурой

MSGSTRUCT STRUC

```
MSHWND       DD  ?   ;идентификатор окна,
                     ;получающего сообщение
MSMESSAGE    DD  ?   ;идентификатор сообщения
MSWPARAM     DD  ?   ;доп. информация о сообщении
MSLPARAM     DD  ?   ;доп. информация о сообщении
MSTIME       DD  ?   ;время посылки сообщения
MSPT         DD  ?   ;положение курсора, во время
                     ;посылки сообщения
MSGSTRUCT ENDS
```

Поле MSHWND содержит значение дескриптора окна, которому предназначено сообщение. Это дескриптор, возвращаемый функцией CreateWindowExA, однозначно идентифицирует окно в системе. приложение обычно имеет несколько окон, поэтому значение в поле MSHWND помогает приложению идентифицировать нужное окно.

В поле MSMESSAGE Windows помещает 32-разрядную константу – идентификатор сообщения, однозначно идентифицирующий тип сообщения. Все эти константы имеют символические имена, начинающиеся с префикса WM_ (Window Message). В программе на языке C/C++ эти константы используются в оконной функции оператором switch для принятия решения о том, какая из его ветвей будет исполняться. В программе на языке ассемблера этот оператор моделируется командами условного и безусловного переходов, а также командой cmp, в качестве второго операнда которой и выступает константа, обозначающая определенный тип сообщения.

Поля MSLPARAM и MSWPARAM предназначены для того, чтобы система Windows могла разместить в них дополнительную информацию о сообщении, необходимую для ее правильной обработки. Эти поля,

например, используются при обработке сообщений о выборе пунктов меню или о нажатии клавиш.

В поле MSTIME Windows записывает информацию о времени, когда сообщение было помещено в очередь сообщений.

Поле MSPT содержит координаты указателя мыши в момент помещения сообщения в очередь.

Функция GetMessage(var Msg: TMsg; Wnd: HWnd; MsgFilterMin, MsgFilterMax: Word): Bool считывает сообщение, в рамках диапазона фильтрации, из очереди сообщений прикладной задачи. Функция оставляет управление другим прикладным задачам, если сообщений нет или если следующим сообщением является wm_Paint или wm_Timer.

Параметры функции GetMessageA:

Msg – принимающая структура MSG;

Wnd – дескриптор окна, сообщения для которого должны будут выбираться функцией GetMessageA, или 0 для всех окон в прикладной задаче;

MsgFilterMin – задает минимальное значение параметра MSMESSAGE, нуль в случае отсутствия фильтрации, или wm_KeyFirst только для клавиатуры, или wm_MouseFirst только для мыши;

MsgFilterMax – задает максимальное значение параметра MSMESSAGE, нуль в случае отсутствия фильтрации, или wm_KeyLast только для клавиатуры, или wm_MouseLast только для мыши.

Функция GetMessageA выполняет следующие действия.

1. Постоянно просматривает очередь сообщений.

2. Выбирает сообщения, удовлетворяющие заданным в функции параметрам.

3. Заносит информацию о сообщении в экземпляр структуры MSG.

4. Передает управление в цикл обработки сообщений.

Цикл обработки сообщений состоит всего из двух функций: TranslateMessage и DispatchMessageA. Эти функции имеют единственный параметр – указатель на экземпляр структуры MSG, предварительно заполненный информацией о сообщении функцией GetMessageA.

Функция TranslateMessage предназначена для обнаружения сообщений от клавиатуры для данного приложения. Если приложение самостоятельно не обрабатывает ввод с клавиатуры, то эти сообщения передаются для обработки обратно Windows. Данная функция переводит комбинации wm_KeyDown/Up в wm_Char или wm_DeadChar и комбинации wm_SysKeyDown/Up в wm_SysChar или wm_SysDeadChar и направляет символьное сообщение в очередь прикладной задачи.

Функция DispatchMessageA предназначена для передачи сообщения оконной процедуре. Такая передача производится не напрямую, так как сама DispatchMessageA ничего не знает о месторасположении оконной процедуры, а косвенно – посредством системы Windows. Это делается следующим образом.

1. Функция DispatchMessageA возвращает сообщение операционной системе.

2. Windows, используя описание класса окна, передает сообщение нужной оконной процедуре приложения.

3. После обработки сообщения оконной процедурой управление возвращается операционной системе.

4. Windows передает управление функции DispatchMessageA.

5. DispatchMessageA завершает свое выполнение.

Так как вызов функции DispatchMessageA является последним в цикле, то управление опять передается функции GetMessageA, которая

выбирает очередное сообщение и, если оно удовлетворяет параметрам, заданным при вызове функции, выполняет тело цикла. Цикл обработки сообщений выполняется до тех пор, пока не приходит сообщение wm_Quit. Получение этого сообщения – единственное условие, при котором программа может выйти из цикла обработки сообщений.

Для завершения работы приложения достаточно использовать функцию ExitProcess@4 с нулевым параметром.

Обработка сообщений в оконной процедуре

Приложение может иметь несколько оконных процедур. Их количество определяется количеством классов окон, зарегистрированных в системе функцией RegisterClassA.

Когда для окна Windows-приложения появляется сообщение, операционная система Windows производит вызов соответствующей оконной процедуры. Сообщения, в зависимости от источника их появления в оконной процедуре, могут быть двух типов: синхронные и асинхронные. К синхронным сообщениям относятся те сообщения, которые помещаются в очередь сообщений приложения и ждут момента, когда они будут выбраны функцией GetMessage. После этого поступившие сообщения попадают в оконную процедуру, где и производится их обработка. Асинхронные сообщения попадают в оконную процедуру в экстренном порядке, минуя все очереди. Асинхронные сообщения, в частности, инициируются некоторыми функциями Win32 API, такими как CreateWindow или UpdateWindow.

Извлечение синхронного сообщения производится функцией GetMessage с последующей передачей обратно в Windows функцией DispatchMessage. Асинхронное сообщение, независимо от источника, кото-

рый инициирует его появление, сначала попадает в Windows и затем – в нужную оконную процедуру.

Windows требует, чтобы оконная процедура сохраняла значения регистров EBI, EDI и ESI. По завершении работы оконная процедура формирует значение в регистре EAX. Если сообщение обрабатывалось в оконной функции, то в EAX необходимо поместить нулевое значение. Если обработка осуществлялась по умолчанию, т.е. функцией DefWindowProc, то в EAX уже сформировано возвращаемое значение, и именно его нужно возвратить в качестве результата работы оконной процедуры.

В приложении А представлен код оконного приложения, которое обрабатывает нажатие левой и правой кнопки мыши.

Методика и порядок выполнения работы

1. Изучить принципы построения оконных приложений на языке ассемблера, взаимодействия приложения с операционной системой Windows.

2. Используя пакет MASM32 произвести компиляцию кода из приложения А. Исследовать полученный исполняемый файл под отладчиком.

3. Разработать программу в соответствии с индивидуальным заданием (Приложение Б) и защитить лабораторную работу преподавателю.

Контрольные вопросы

1. Из каких частей состоит оконное Windows-приложение?

2. Как производится вызов функций Win32 API?

3. Для чего требуется регистрировать класс?

4. Какие функции используются в стартовом коде?

5. Что выполняет петля обработки сообщений?

6. Как реализуется взаимодействие операционной системы Windows с оконным приложением?

7. Что выполняет оконная процедура?

8. Как реализуется передача сообщения оконной процедуре?

9. Какие типы сообщений бывают?

10. Как вывести окно приложения на экран?

Лабораторная работа № 2

Оконное приложение с управляющими элементами

Цель работы: Получить навык создания оконных приложений Windows с дочерними окнами.

Краткие сведения из теории

Кнопки, списки, окна редактирования и другие элементы управления являются дочерними окнами главного окна приложения. Эти элементы, по сути, также являются окнами, но обладающими особыми свойствами. События, происходящие с этими элементами (и самим окном), приводят к приходу сообщений в оконную процедуру. Поэтому необходимо в оконную процедуру включить обработку событий с содержимым окна.

Сообщение WM_COMMAND генерируется системой, когда что-то происходит с управляющими элементами окна. В этом случае по значению параметров можно определить, какой это элемент и что с ним произошло (LPARAM – дескриптор элемента, старшее слово WPARAM – событие, младшее слово WPARAM – обычно идентификатор ресурса). Например, событие «кнопка нажата» может быть определено следующим образом. В начале производится проверка сообщения WM_COMMAND, а затем проверяем LPARAM – здесь хранится дескриптор (уникальный номер) окна (кнопка создается как окно).

В качестве примера рассмотрим приложение, интерфейс которого включает две кнопки и окно редактирования текста. При нажатии кнопки «Выполнить» создается окно-сообщение с кнопкой «ОК» и текстом из окна редактирования. По нажатию кнопки «Выход» происходит закрытие приложения.

Вначале программы опишем константы, которые используются для настройки окон и обработки событий.

```
.386P
.MODEL FLAT, stdcall
; сообщение приходит при закрытии окна
WM_DESTROY                  equ 2
; сообщение приходит при создании окна
WM_CREATE                   equ 1
;сообщение приходит при нажатии кнопки
WM_COMMAND                  equ 111h
```

Свойства главного окна:

```
CS_VREDRAW                  equ 1h
CS_HREDRAW                  equ 2h
CS_GLOBALCLASS              equ 4000h
WS_OVERLAPPEDWINDOW         equ 000CF0000H
style equ CS_HREDRAW+CS_VREDRAW+CS_GLOBALCLASS
```

Свойства кнопки:

```
BS_DEFPUSHBUTTON            equ 1h
WS_VISIBLE                  equ 10000000h
WS_CHILD                    equ 40000000h
STYLBTN equ WS_CHILD + BS_DEFPUSHBUTTON + WS_VISIBLE
```

Свойства кнопки, которая будет создано как окно – это наиболее типичное сочетание свойств, но не единственное. Например, для того чтобы кнопка содержала иконку, то необходимым условием для этого будет свойство BS_ICON (или BS_BITMAP).

Свойства окна редактирования:

```
WS_BORDER                equ 800000h
ES_AUTOHSCROLL           equ 80h
STYLEDT equ WS_CHILD + WS_VISIBLE + WS_BORDER + ES_AUTOHSCROLL
```

Также как и свойства кнопки, свойства окна редактирования могут быть дополнены другими константами. Например, константа ES_AUTOHSCROLL позволит избавиться от ограничения на число вводимых символов, которое определяется размером окна редактирования.

Зададим идентификаторы иконки и курсора:

```
IDI_APPLICATION          equ 32512
IDC_CROSS                equ 32515
```

Режим показа окна установим нормальным:

```
SW_SHOWNORMAL            equ 1
```

Определим прототипы внешних функций

```
EXTERN    MessageBoxA@16:  NEAR
EXTERN    CreateWindowExA@48:  NEAR
EXTERN    DefWindowProcA@16:  NEAR
EXTERN    DispatchMessageA@4:  NEAR
EXTERN    ExitProcess@4: NEAR
EXTERN    GetMessageA@16:  NEAR
EXTERN    GetModuleHandleA@4:  NEAR
EXTERN    LoadCursorA@8:  NEAR
EXTERN    LoadIconA@8:  NEAR
EXTERN    PostQuitMessage@4:  NEAR
EXTERN    RegisterClassA@4: NEAR
EXTERN    ShowWindow@8: NEAR
EXTERN    TranslateMessage@4:  NEAR
EXTERN    UpdateWindow@4:  NEAR
EXTERN    GetWindowTextA@12:  NEAR
```

Директивы компоновщику для подключения библиотек, структуры сообщения и класса аналогичны директивам и структурам Приложения А.

Далее определяем сегмент данных. В сегменте данных необходимо зарезервировать три двойных слова для хранения дескрипторов двух кнопок и окна редактирования, которые будут получены после их создания функцией CreateWindowExA@48. Также необходимо определить уникальные имена классов кнопок и окна редактирования. Для хранения введенных символов в сегменте данных зарезервируем 50 байт (buf).

```
_DATA SEGMENT  DWORD PUBLIC  USE32 'DATA'
      NEWHWND            DD 0            ;дескриптор главного окна
      MSG                MSGSTRUCT  <?>
      WC                 WNDCLASS  <?>
      HINST              DD 0            ;дескриптор приложения
      TITLENAME          DB 'Простой пример 32-битного приложения', 0
      CLASSNAME          DB 'CLASS32', 0
      ButtonClassName    DB "button",  0  ;имя класса кнопки
      ButtonText1        DB "Выполнить", 0
      ButtonText2        DB "Выход",  0
      EditClassName      DB "edit", 0     ;имя класса окна редактирования
      hwndButton1        DD ?        ;дескриптор первой кнопки
      hwndButton2        DD ?        ;дескриптор второй кнопки
      hwndEdit1          DD ?        ;дескриптор окна редактирования
      CAP                DB 'Сообщение', 0
      MES1               DB 'Выход из программы', 0
      buf                DB 50 DUP(0)          ;буфер
_DATA ENDS
```

Стартовый код, регистрация класса главного окна и создание последнего, а также цикл обработки сообщений в данном примере ничем не отличается от примера в Приложении А. Рассмотрим оконную процедуру.

Начало оконной процедуры включает проверку сообщений:

```
WNDPROC  PROC
      PUSH EBP
```

```
        MOV EBP, ESP
        PUSH EBX
        PUSH ESI
        PUSH EDI
CMP DWORD PTR [EBP+0CH], WM_DESTROY
        JE  WMDESTROY
CMP DWORD PTR [EBP+0CH], WM_CREATE
        JE  WMCREATE
CMP DWORD PTR [EBP+0CH] ,WM_COMMAND
        JE  WMCOMMAND
        JMP DEFWNDPROC
```

По приходу сообщения WM_CREATE необходимо создать дочерние окна (две кнопки и окно редактирования текста). Следующий код в оконной процедуре создает дочерние окна и сохраняет в сегменте данных их дескрипторы, которые потребуется при обработке сообщений WM_COMMAND.

```
WMCREATE:
;-------создаем кнопку "Выполнить"
        PUSH 0
        PUSH [HINST]
        PUSH 0
        PUSH DWORD PTR [EBP+08H]
        PUSH 20 ;DY – высота окна
        PUSH 100 ;DX – ширина окна
        PUSH 10 ;Y – координата левого верхнего угла
        PUSH 10 ;X – координата левого верхнего угла
        PUSH STYLBTN
        PUSH OFFSET ButtonText1  ; имя окна
        PUSH OFFSET ButtonClassName  ; имя класса
        PUSH 0
        CALL CreateWindowExA@48
        mov hwndButton1, EAX
;--------создаем кнопку "Выход"
        PUSH 0
        PUSH [HINST]
```

```
	PUSH 0
	PUSH DWORD PTR [EBP+08H]
	PUSH 20 ; DY – высота окна
	PUSH 100 ; DX – ширина окна
	PUSH 40 ; Y – кордината левого верхнего угла
	PUSH 10 ; X – кордината левого верхнего угла
	PUSH STYLBTN
	PUSH OFFSET ButtonText2  ; имя окна
	PUSH OFFSET ButtonClassName  ; имя класса
	PUSH 0
	CALL CreateWindowExA@48
	mov hwndButton2, EAX
;--------создаем окно редактирования текста
	PUSH 0
	PUSH [HINST]
	PUSH 0
	PUSH DWORD PTR [EBP+08H]
	PUSH 20 ; DY – высота окна
	PUSH 200 ; DX – ширина окна
	PUSH 10 ; Y – координата левого верхнего угла
	PUSH 120 ; X – координата левого верхнего угла
	PUSH STYLEDT
	PUSH 0
	PUSH OFFSET EditClassName ; имя класса
	PUSH 0
	CALL CreateWindowExA@48
	mov hwndEdit1, EAX
	MOV EAX, 0
	JMP FINISH
```

Обработка сообщений от управляющих элементов включает проверку дескриптора элемента и действия в случае совпадения.

```
WMCOMMAND:
	mov eax, hwndButton1
	CMP DWORD PTR [EBP+14H], eax
	JNE  NEXTE
	PUSH 50
	PUSH OFFSET buf
```

```
        PUSH hwndEdit1
        CALL GetWindowTextA@12
        PUSH 0
        PUSH OFFSET CAP
        PUSH OFFSET buf
        PUSH NEWHWND  ; дескриптор окна
        CALL MessageBoxA@16
NEXTE:
        mov eax, hwndButton2
        CMP DWORD PTR [EBP+14H], eax
        JE  WMDESTROY
        MOV EAX, 0
        JMP FINISH
```

Функция GetWindowTextA@12 сохраняет в указанном буфере текст из окна редактирования.

Заключительный код оконной процедуры совпадает с кодом примера Приложения А.

```
DEFWNDPROC:
        PUSH DWORD PTR [EBP+14H]
        PUSH DWORD PTR [EBP+10H]
        PUSH DWORD PTR [EBP+0CH]
        PUSH DWORD PTR [EBP+08H]
        CALL DefWindowProcA@16
        JMP FINISH
WMDESTROY:
        PUSH 0         ; MB_OK
        PUSH OFFSET CAP
        PUSH OFFSET MES1
        PUSH DWORD PTR [EBP+08H] ; дескриптор окна
        CALL MessageBoxA@16
        PUSH 0
        CALL PostQuitMessage@4  ; сообщение WM_QUIT
        MOV EAX, 0
FINISH:
        POP EDI
        POP ESI
```

```
        POP EBX
        POP EBP
        RET 16
WNDPROC ENDP
_TEXT ENDS
         END START
```

Сочетание различных констант в свойствах дочерних окон позволяет получить различные управляющие элементы. Например, список – это окно со стилем

STYLLST equ WS_THICKFRAME + WS_CHILD + WS_VISIBLE + WS_BORDER + WS_TABSTOP + WS_VSCROLL + LBS_NOTIFY

Ход выполнения работы

1. Ознакомиться с константами, задающими стиль дочерних окон.

2. Создать оконное приложение с тремя кнопками и двумя окнами редактирования. Одна из трех кнопок реализует выход из приложения. Остальные две кнопки – действия, соответствующие индивидуальному заданию.

3. Защитить лабораторную работу преподавателю.

Контрольные вопросы

1. Какие управляющие элементы можно создать?

2. Как создаются кнопки на главном окне?

3. Обработка сообщений от управляющих элементов.

4. Как получить дескриптор дочернего окна, и для чего он нужен?

5. Какие поля структуры сообщения используются для управления?

Таблица 2.1 – Задания на лабораторную работу № 2

№	Кнопка № 1	Кнопка № 2
1	**2**	**3**
1	Создать каталог с именем из Edit1	Создать в каталоге с именем из Edit1 файл с расширением .txt и именем из Edit2
2	Создать файл с именем из Edit1	Записать в созданный файл строку из Edit2
3	Открыть файл с именем из Edit1	Вывести в Edit2 второе слово первой строки из открытого файла
4	Записать в файл с именем из Edit1 строку из буфера	Вывести в Edit2 размер файла с именем из Edit1
5	Вывести в Edit1 свободное пространство на диске C:\	Создать файл с именем из Edit2 в корне диска C:\
6	Удалить файл с именем из Edit1	Определить набор символов файла (ANSI или OEM) с именем из Edit2
7	Определить является ли файл с именем из Edit1 исполняемым	Вывести в Edit2 для какой подсистемы файл с именем из Edit1 является исполняемым – Win32, MS DOS, OS/2, UNIX (POSIX) и т.д.
8	Вывести в Edit1 текущий каталог	Определить и вывести тип диска из Edit2 с помощью функции MessageBoxA: съемный, фиксированный, CD-ROM, электронный или сетевой
9	Получить атрибуты файла именем из Edit1 и вывести их с помощью функции MessageBoxA	Вывести в Edit2 тип файла, указанного в Edit1
10	Вывести с помощью функции MessageBoxA полный путь и имя для указанного в Edit1 файла	Вывести в Edit2 доступные в настоящее время дисководы
11	Блокировать файл с именем из Edit1	Переименовать файл с именем из Edit2 в файл с именем из Edit1
12	Удалить существующий каталог с именем из Edit1	Изменить атрибуты файла с именем из Edit2
13	Закрыть открытый дескриптор файла с именем из Edit1	Вывести в Edit2 последнюю строку файла с именем из Edit1

1	**2**	**3**
14	Найти файл с именем из Edit1. Результат поиска вывести на экран с помощью функции MessageBoxA	Вывести в Edit2 псевдоним файла с именем из Edit1
15	Получить атрибуты каталога с именем из Edit1. Результат вывести на экран с помощью функции MessageBoxA	Получить информацию об изменениях в пределах каталога, имя которого указано в Edit2
16	Установить текущим каталог с именем из Edit1	Записать в существующий файл с именем из Edit2 имя текущего каталога в первую строку без удаления содержимого
17	Вывести на экран с помощью функции MessageBoxA время создания файла с именем из Edit1	Скопировать файл с именем из Edit1 в файл с именем из Edit2
18	Создать файл с именем из Edit1 и записать в него доступные в настоящее время дисководы	Перемесить файл с именем из Edit1 в файл с именем из Edit2
19	Вывести в Edit1 имя CD-ROM, если он имеется	Открыть файл с именем из Edit2
20	Определить наличие флеш-диска в системе и вывести его имя в Edit1	Создать на флеш-диске с именем из Edit1 каталог с именем из Edit2
21	Вывести в Edit1 текущую дату	Создать файл с именем из Edit2 и записать в него время создания
22	Изменить текущее время на время из Edit1	Создать каталог с именем из Edit2 в папке «Мои документы», предварительно определив тип операционной системы
23	Вывести в Edit1 тип операционной системы	Создать файл с именем из Edit2 и записать в него объем диска C:\ в байтах
24	Найти на диске файл с именем из Edit1 и удалить его	Создать файл с именем из Edit1 и записать в него строку из Edit2
25	Записать в файл, имя которого указано в Edit1, свои ФИО в последнюю строку	Создать файл с именем из Edit2 и записать в него первые две строки из файла с именем из Edit1

Лабораторная работа № 3

Разработка программ, использующих ресурсы

Цель работы: Получить навык создания оконных приложений Windows, использующих ресурсы.

Краткие сведения из теории

В операционную систему Windows введено понятие ресурса. Ресурс представляет собой некий визуальный элемент с заданными свойствами, хранящийся в исполняемом файле отдельно от кода и данных, который может отображаться специальными функциями.

Использование ресурсов дает две вполне определенные выгоды:

1. ресурсы загружаются в память лишь при обращении к ним, т.е. реализуется экономия памяти;
2. свойства ресурсов поддерживаются системой автоматически, не требуя от программиста написания дополнительного кода.

Описание ресурсов хранится отдельно от программы в текстовом файле (*.rc) и компилируется (*.res) специальным транслятором ресурсов. В исполняемый файл ресурсы включаются компоновщиком. Транслятором ресурсов в пакете MASM32 является RC.EXE.

Наиболее употребляемые ресурсы:

1. иконки;
2. курсоры;
3. битовая картинка;
4. строка;
5. диалоговое окно;
6. меню;
7. акселераторы.

Такой ресурс, как диалоговое окно, может содержать в себе управляющие элементы, которые также должны быть описаны, но в рамках описания окна.

Чтобы добавить этот файл в программу, его надо скомпилировать и указать имя скомпилированного *.RES-файла для компоновщика:

```
ml /c /coff /Cp lab3.asm
rc /r winmenu.rc
link lab3.obj winmenu.res  /subsystem:windows
```

В качестве примера рассмотрим файл ресурсов (winmenu.rc), который содержит:

```
#define ZZZ_TEST  0
#define ZZZ_OPEN  1
#define ZZZ_SAVE  2
#define ZZZ_EXIT  3
ZZZ_Menu  MENU {
   POPUP "&File" {
      MENUITEM  "&Open",ZZZ_OPEN
      MENUITEM  "&Save", ZZZ_SAVE
      MENUITEM  SEPARATOR
      MENUITEM  "E&xit",ZZZ_EXIT
   }
   MENUITEM  "&Edit",ZZZ_TEST
}
```

Рассмотрим код программы, использующей меню (lab3.asm).

Прежде необходимо добавить константы меню:

```
ZZZ_TEST     equ     0
ZZZ_OPEN     equ     1
ZZZ_SAVE     equ     2
ZZZ_EXIT     equ     3
```

Сообщения от нашего меню должны совпадать с определениями из winmenu.rc. Кроме того, в данном примере их номера важны, потому что они используются как индекс для таблицы переходов к обработчикам.

В сегмент данных необходимо добавить имя класса меню:

```
menu_name    db    "ZZZ_Menu",0;
```

и выделить четыре байта для дескриптора меню:

```
menu_hinst    dd    0
```

После регистрации класса окна необходимо загрузить меню, для чего служит следующий код:

```
push    offset  menu_name
push    [HINST]
call       LoadMenuA@8
mov        menu_hinst,  eax
```

Функция LoadMenuA@8 получает указатель на меню из файла ресурсов, который записывается в EAX.

Далее при создании окна необходимо в параметр Menu загрузить идентификатор разработанного меню.

Теперь перейдем к процедуре окна. Как и от любого другого управляющего элемента при выборе определенного элемента меню приходит сообщение WM_COMMAND, которое в LPARAM будет содержать дескриптор меню, а в WPARAM – идентификатор пункта меню, которые ранее мы задали как константы (ZZZ_TEST, ZZZ_OPEN, ZZZ_SAVE, ZZZ_EXIT). Следовательно, для обработки событий меню необходимо обнаружить его дескриптор и по коду пункта выполнить соответствующее действие. Например, обработка пункта выхода из приложения «Exit»:

```
mov eax, menu_hinst
CMP dword ptr [ebp+14h],eax
jne go_away
cmp dword ptr [ebp+10h],ZZZ_EXIT
je WMDESTROY
```

На рисунке 3.1 представлено меню рассмотренного примера.

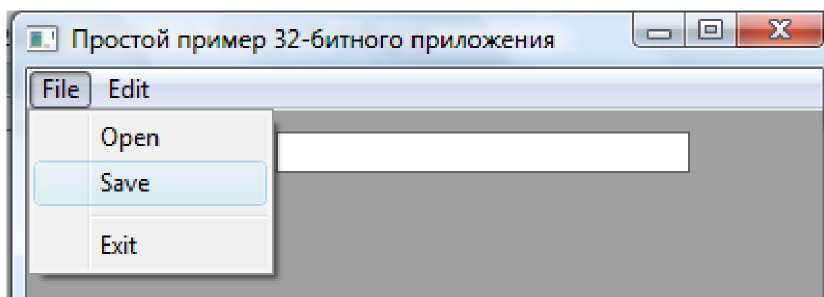


Рисунок 3.1 – Одноуровневое меню

Рассмотрим структуру меню с несколькими уровнями. На рисунке 3.2 представлена иерархическая структура меню.

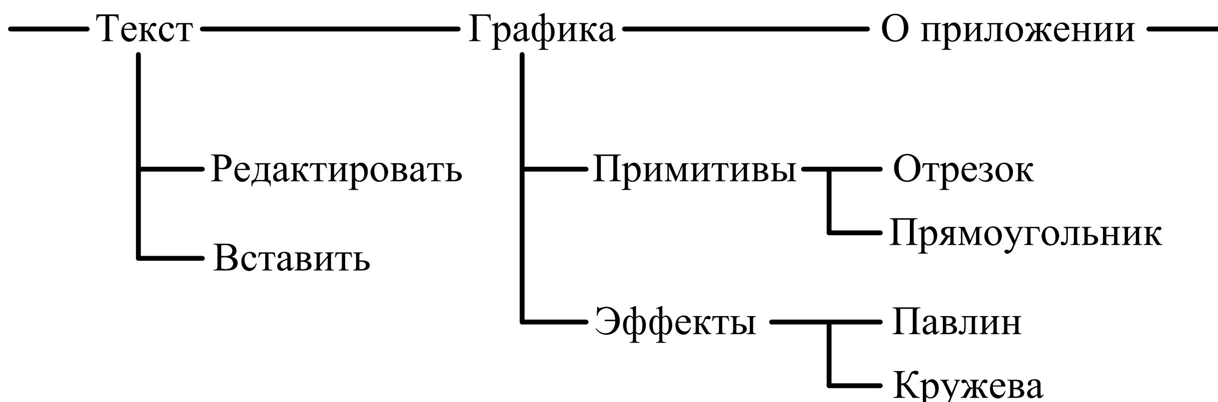


Рисунок 3.2 – Структура меню с двумя уровнями

Далее приведем текст файла ресурса меню рисунка 3.2.

```
#define IDM_DRAWTEXT  0
#define IDM_TEXTOUT  1
#define IDM_LENGTH  2
#define IDM_RECTANGLE  3
#define IDM_PEACOCK  4
#define IDM_LACES  5
#define IDM_ABOUT  6
MYMENU  MENU
{
POPUP  “&Текст”
            {
            MENUITEM  “&Редактировать”, IDM_DRAWTEXT
            MENUITEM  “&Вставить”, IDM_TEXTOUT
            }
POPUP  “&Графика”
```

```
        {
        POPUP “&Примитивы”
                {
                MENUITEM  “&Отрезок”, IDM_LENGTH
                MENUITEM  “&Прямоугольник”, IDM_RECTANGLE
                }
        POPUP “&”Эффекты”
                {
                MENUITEM  “&Павлин”, IDM_PEACOCK
                MENUITEM  “&Кружева”, IDM_LACES
                }
        }
MENUITEM  “&О приложении”, IDM_ABOUT
}
```

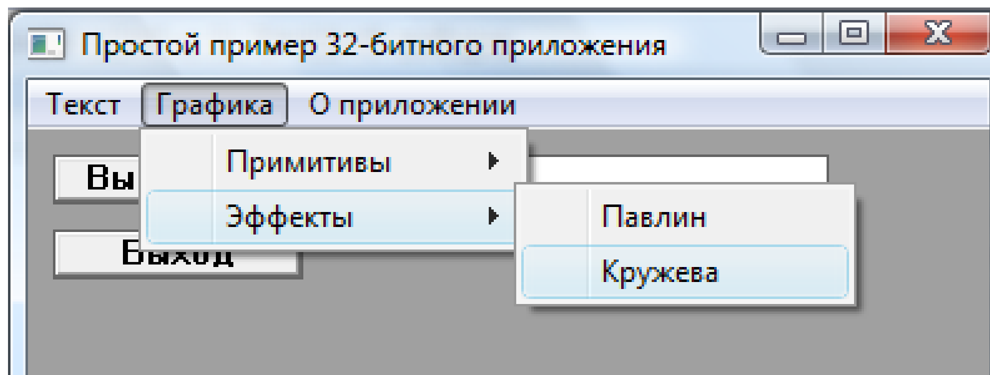


Рисунок 3.3 – Двухуровневое меню

Ход выполнения работы

1. Ознакомиться с принципами организации меню.

2. К приложению, разработанному во второй лабораторной работе, добавить меню, изображенное на рисунке 4. При выборе пункта «Очистить» должна произойти очистка поля для редактирования текста. Пункты «Действие 1» и «Действие 2» соответствуют заданиям на лабораторную работу № 2.

3. Защитить лабораторную работу преподавателю.

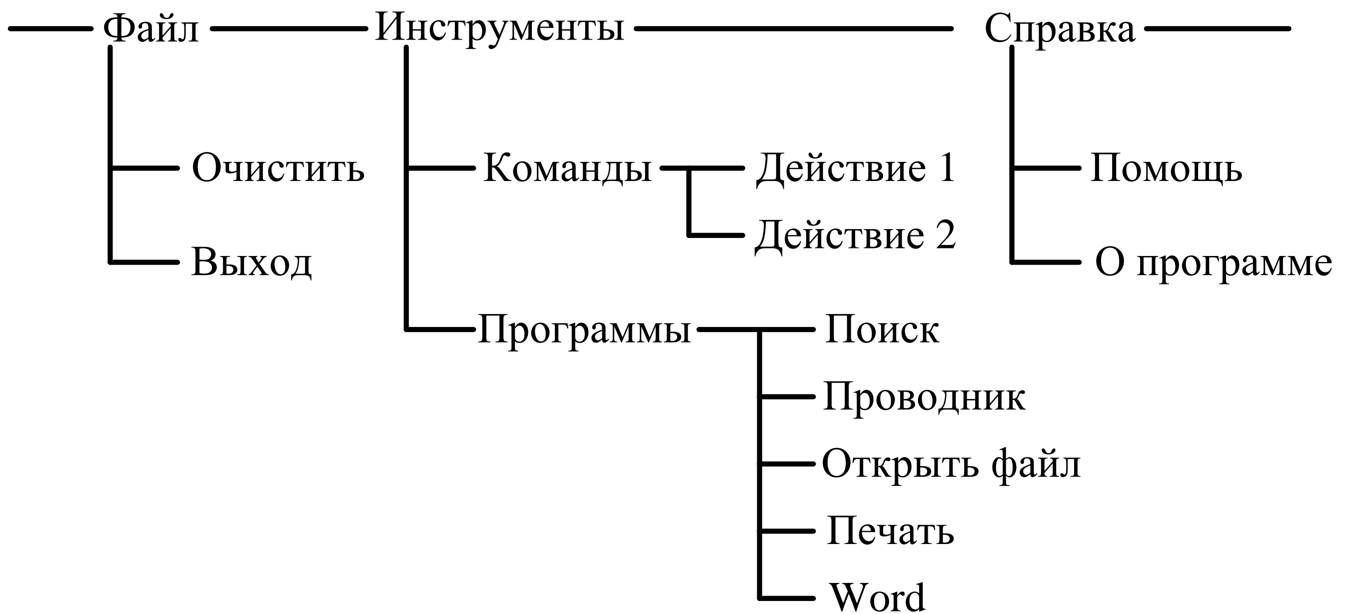


Рисунок 4 – Структура меню задания на лабораторную работу

Программы «Поиск» и «Проводник» можно вызвать функцией ShellExecute. Функция ShellExecute из библиотеки Shell32.dll выполняет операцию над указанным файлом. Вот её прототип:

HINSTANCE ShellExecute (HWND *hwnd*, LPCTSTR *lpOperation*, LPCTSTR *lpFile*, LPCTSTR *lpParameters*, LPCTSTR *lpDirectory*, INT *nShowCmd*).

В случае успешного завершения функция возвращает значение, большее 32. При возникновении ошибки функция вернёт одно из следующих значений:

Код ошибки	Описание
0	Недостаточно памяти или ресурсов Windows
2	Указанный файл не найден
3	Указанный путь не существует
5	Операционная система не имеет доступа к указанному файлу
8	Недостаточно памяти для завершения операции
26	Невозможен совместный доступ к файлу
27	Невозможно загрузить приложение, ассоциированное с типом файла
28	DDE транзакция не может быть завершена из-за истечения времени
29	Неудача при выполнении DDE транзакции
30	DDE транзакция не может быть завершена, так как обрабатываются другие DDE-транзакции
31	Нет никакого приложения, ассоциированного с расширением файла
32	Не найдена указанная DLL-библиотека

Функции передаются следующие параметры:

Параметр	Описание
hwnd	Дескриптор родительского окна. При вызове функции из Visual FoxPro должен быть равен нулю.
Operation	Может принимать одно из следующих значений: "find", "explore", "edit", "open" или "print"
File	Имя файла или папки – в зависимости от значения параметра *Operation*.
Parameters	Список параметров, передаваемых загружаемому приложению
Directory	Путь к файлу, указанному в *File*
ShowCmd	Определяет вид главного окна загружаемого приложения

Если *Operation="find"*, функция выводит диалоговое окно для поиска файлов по условиям. Параметр *File* должен указывать путь к папке, начиная с которой будет выполняться поиск. Остальные параметры не используются.

Если *Operation="explore"*, функция выводит диалоговое окно – список папок. Параметр *File* должен указывать путь к папке, содержимое которой нужно посмотреть. Остальные параметры не используются.

Если *Operation="edit"*, функция открывает файл на редактирование, загружая приложение, ассоциированное с расширением файла. Параметр *Edit* должен содержать имя файла, параметр *Directory* – указывать путь к этому файлу; если параметр *Directory* не используется, то параметр *Edit* должен указывать путь и имя файла.

Если *Operation="open"*, функция выполняет следующие действия: если в *File* указан исполняемый файл (например, типа EXE), то он запускается на выполнение; загружаемой программе передаётся список параметров, указанных в *Parameters*; в противном файл открывается на редактирование.

Если *Operation="print"*, то выполняется печать файла на принтере (фактически загружается ассоциированное с расширением файла приложение, которое и печатает документ).

Параметр *ShowCmd* может принимать значения от 0 до 10, реальный интерес представляют значения, перечисленные в таблице:

ShowCmd	Описание
0	Скрывает окно загружаемого приложения и активизирует другое окно.
1	Отображает главное окно приложения и делает его активным. Если окно приложения минимизировано или максимизировано, Windows восстанавливает его первоначальный размер и позицию.
2	Окно загружаемого приложения минимизировано.
3	Раскрывает окно приложения на весь экран и делает его активным.
4	Отображает окно приложения в его последних сохранённых размерах, но не делает его активным.

Примеры использования функции в вашем приложении.

В следующем примере запускается приложение Notepad (блокнот); окно приложения распахивается на весь экран и становится активным:
nReturn = ShellExecute(0,'open','c:\Windows\Notepad.exe',NULL,NULL,3).

В следующем примере загружается приложение MS Word для редактирования файла MyDocument.doc, расположенного в папке c:\MyDocs:
nReturn = ShellExecute(0,'open','MyDocument.doc', NULL,'c:\MyDocs',1).

И последний пример, в котором при помощи MS Word выполняется печать файла MyDocument.doc. MS Word загружается в скрытом (Hide) режиме (окно не выводится, индикатор в панели "Пуск" не отображается). Документ распечатывается на принтере, используемом по умолчанию:
nReturn = ShellExecute(0,'print','c:\MyDocs\MyDocument.doc',
NULL,NULL,0).

Контрольные вопросы

1. Что такое ресурс и какие ресурсы Вам известны?

2. В чем заключается преимущество использования ресурсов?

3. Как создать двух и трехуровневое меню?

4. Как обрабатываются сообщения от меню?

5. На каком этапе работы программы загружается меню?

6. Как запустить файл с расширением exe с помощью функции Win32 API?

7. Как меню прикрепляется к главному окну?

Лабораторная работа № 4

Разработка DLL-библиотеки

Цель работы: Получить навык создания библиотек динамической компоновки.

Краткие сведения из теории

Использование динамических библиотек (по-другому – библиотек динамической компоновки) – это способ осуществления модульности в период выполнения программы. Динамическая библиотека (Dynamic Link Library – DLL) позволяет упростить разработку программного обеспечения. Вместо того чтобы каждый раз перекомпилировать огромные EXE-программы, достаточно перекомпилировать лишь отдельный динамический модуль. Кроме того, доступ к динамической библиотеке возможен сразу из нескольких исполняемых модулей, что делает многозадачность более гибкой. Использование динамической библиотеки экономит дисковое пространство, т.к. представленная в библиотеке процедура содержится лишь один раз, в отличие от процедур, помещаемых в модули из статических библиотек.

С точки зрения программирования на ассемблере DLL – это самый обычный исполнимый файл формата PE, отличающийся только тем, что

при входе в него в стеке находятся три параметра (идентификатор DLL-модуля, причина вызова процедуры и зарезервированный параметр), которые надо удалить, например командой ret 12. Кроме этой процедуры в DLL входят и другие, часть которых можно вызывать из других программ. Список этих экспортируемых процедур должен быть задан во время компиляции DLL, и поэтому команды для компиляции будут отличаться от обычных.

Рассмотрим различные варианты связывания при трансляции. Во время трансляции связываются имена, указанные в программе как внешние, (EXTERN) с соответствующими именами из библиотек, которые указываются при помощи директивы IMPORTLIB. Такое связывание называется ранним (или статическим). Напротив, в случае с динамической библиотекой связывание происходит во время выполнения модуля. Такое связывание называется поздним (или динамическим). При этом позднее связывание может происходить в автоматическом режиме в начале запуска программы и при помощи специальных API-функций (см. ниже), по желанию программиста. При этом говорят о явном и неявном связывании.

Динамическая библиотека может содержать также ресурсы. Так, файлы шрифтов представляют собой динамические библиотеки, единственным содержимым которых являются ресурсы. Динамическая библиотека как бы становится продолжением программы, загружаясь в адресное пространство процесса. Соответственно, данные процесса доступны из динамической библиотеки, и, наоборот, данные динамической библиотеки доступны для процесса.

В любой динамической библиотеке следует определить точку входа (процедура входа). По умолчанию за точку входа принимают метку, указываемую за директивой END (например, END START). При загрузке ди-

намической библиотеки и выгрузке динамической библиотеки автоматически вызывается процедура входа. Заметим при этом, что каким бы способом ни была загружена динамическая библиотека (явно или неявно), выгрузка динамической библиотеки из памяти будет происходить автоматически при закрытии процесса или потока. В принципе, процедура входа может быть использована для некоторой начальной инициализации переменных. Довольно часто эта процедура остается пустой. При вызове процедуры входа в нее помещаются три параметра:

1. идентификатор DLL-модуля;
2. причина вызова;
3. резерв.

Рассмотрим подробнее второй параметр процедуры входа. Вот четыре возможных значения этого параметра:

DLL_PROCESS_DETACH equ 0

DLL_PROCESS_ATTACH equ 1

DLL_THREAD_ATTACH equ 2

DLL_THREAD_DETACH equ 3

DLL_PROCESS_ATTACH – сообщает, что динамическая библиотека загружена в адресное пространство вызывающего процесса.

DLL_THREAD_ATTACH – сообщает, что текущий процесс создает новый поток. Такое сообщение посылается всем динамическим библиотекам, загруженным к этому времени процессом.

DLL_PROCESS_DETACH – сообщает, что динамическая библиотека выгружается из адресного пространства процесса.

DLL_THREAD_DETACH – сообщает, что некий поток, созданный данным процессом, в адресное пространство которого загружена данная динамическая библиотека, уничтожается.

Лабораторная работа включает следующие четыре этапа.

1. Разработка текста DLL-библиотеки.

2. Трансляция и компоновка исходного текста DLL-библиотеки.

3. Сборка приложения с использованием DLL-библиотеки.

4. Проверка работоспособности приложения с использованием DLL-библиотеки.

Разработка текста DLL-библиотеки

В качестве примера рассмотрим библиотеку, содержащую функцию вычисления остатка от целочисленного деления. Функции необходимо передать два параметра (делимое и делитель) и возвращает один параметр в регистре EAX. Программа, которая использует данную функцию, будет выводить результат в стандартном окне-сообщении.

Ниже приводится код DDL-библиотеки.

```
.386P
.model flat, stdcall
public mod_asm        ;разработанная функция
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
; [EBP+10H]    резервный параметр
; [EBP+0CH]    причина вызова
; [EBP+8]      идентификатор DLL-модуля
_start@12:
MOV EAX,1
RET 12
mod_asm proc EXPORT
pop ecx               ; обратный адрес в ecx
pop eax               ; делимое
pop ebx               ; делитель
```

```
push ecx            ; обратный адрес вернуть в стек для RET
xor edx, edx
div ebx
mov eax, edx
ret
mod_asm endp
_TEXT ends
end _start@12
```

Обратите внимание, что за процедурой, вызываемой из другого модуля, мы указали ключевое слово EXPORT. Это слово необходимо для правильной трансляции в MASM.

Трансляция и компоновка исходного текста DLL-библиотеки

```
ml /c /coff /DMASM mod.asm
link /subsystem:windows /DLL /ENTRY:_start@12 mod.obj
```

MASM помещает в динамическую библиотеку вместо mod_asm имя _mod_asm@0, которое нужно учесть в программе.

В результате трансляции получены mod.dll, mod.lib, mod.exp.

Сборка приложения с использованием DLL-библиотеки

Рассмотрим приложение, которое использует явное связывание. Библиотека должна быть вначале загружена при помощи функции LoadLibrary. Затем определяется адрес процедуры с помощью функции GetProcAddress, после чего можно осуществлять вызов. Также необходимо учесть возможность ошибки при вызове функций LoadLibrary и GetProcAddress. В этой связи укажем, как (в какой последовательности) ищет библиотеку функция LoadLibrary:

1. Поиск в каталоге, откуда была запущена программа.

2. Поиск в текущем каталоге.

3. В системном директории (GetSystemDirectory).

4. В директории Windows (GetWindowsDirectory).

5. В каталогах, указанных в окружении (PATH).

В конце программы мы выгружаем из памяти динамическую библиотеку, что, кстати, могли бы и не делать, т.к. по выходе из программы эта процедура выполняется автоматически.

```
.386P
.model flat, stdcall
EXTERN     GetProcAddress@8:NEAR
EXTERN     LoadLibraryA@4:NEAR
EXTERN     FreeLibrary@4:NEAR
EXTERN     ExitProcess@4:NEAR
EXTERN     MessageBoxA@16:NEAR
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib
_DATA SEGMENT DWORD PUBLIC USE32 'DATA'
TXT   DB 'Ошибка динамической библиотеки',0
MS    DB 'Сообщение',0
LIBR  DB 'mod.dll',0
HLIB  DD ?
TXT1 DB 0, 0
NAMEPROC DB '_mod_asm@0',0
_DATA ENDS
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
START:
; загрузить библиотеку
   PUSH OFFSET LIBR
```

```
    CALL LoadLibraryA@4
    CMP EAX,0
    JE _ERR
    MOV HLIB,EAX
; получить адрес процедуры
    PUSH OFFSET NAMEPROC
    PUSH HLIB
    CALL GetProcAddress@8
    CMP EAX,0
    JNE YES_NAME
; сообщение об ошибке
_ERR:
    PUSH  0
    PUSH  OFFSET MS
    PUSH  OFFSET TXT
    PUSH  0
    CALL  MessageBoxA@16
    JMP  _EXIT
YES_NAME:
    PUSH 7 ; делитель
    PUSH 112 ; делимое
    CALL EAX
; вывод результата деления в сообщении
   xor eax, 30h
   mov byte ptr [TXT1], al
    PUSH  0
    PUSH  OFFSET MS
```

```
    PUSH  OFFSET TXT1
    PUSH  0
    CALL  MessageBoxA@16
; закрыть библиотеку
    PUSH HLIB
    CALL FreeLibrary@4
; библиотека автоматически закрывается также
; при выходе из программы
; выход
_EXIT:
    PUSH 0
    CALL ExitProcess@4
_TEXT ENDS
END START
```

Теперь рассмотрим программу, которая использует неявное связывание. Во-первых, необходимо объявить вызываемую из динамической библиотеки процедуру как внешнюю, а, во-вторых, подключить статическую библиотеку mod.dll.

```
.386P
.model flat, stdcall
includelib mod.lib
EXTERN      mod_asm@0:  NEAR
EXTERN      ExitProcess@4: NEAR
EXTERN      MessageBoxA@16:  NEAR
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib
_DATA SEGMENT DWORD PUBLIC  USE32 'DATA'
```

```
MS     DB 'Сообщение',0
TXT1 DB 0, 0
_DATA ENDS
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
START:
    PUSH 7 ; делитель
    PUSH 112 ; делимое
    CALL mod_asm@0
; вывод результата деления в сообщении
   xor eax, 30h
   mov byte ptr [TXT1], al
    PUSH  0
    PUSH  OFFSET MS
    PUSH  OFFSET TXT1
    PUSH  0
    CALL  MessageBoxA@16
    PUSH 0
    CALL ExitProcess@4
_TEXT ENDS
END START
```

Методика и порядок выполнения работы

1. Изучить принципы разработки библиотек динамической компоновки в операционной системе Windows.

2. Разработать DLL и оконное приложение, использующее функции из разработанной библиотеки. Разработать приложение с явным и приложение с неявным связыванием. Библиотека должна содержать две функции.

3. Защитить лабораторную работу преподавателю.

Таблица 4.1 – Варианты индивидуальных заданий на лабораторную работу № 4

№	Первая функция	Вторая функция
1	**2**	**3**
1	$y1=\begin{cases} a+x, \ if \ x>a \\ 2a-x, \ if \ x\le a \end{cases}$	$y2=\begin{cases} a-x, \ if \ x>a \\ 2ax, \ if \ x\le a \end{cases}$
2	$y1=\begin{cases} x-2, \ if \ x\ge 2 \\ 8, \ if \ x<2 \end{cases}$	$y2=\begin{cases} 4, \ if \ x=0 \\ a-x, \ if \ x\ne 0 \end{cases}$
3	$y1=\begin{cases} x-a, \ if \ x>a \\ 5, \ if \ x\le a \end{cases}$	$y2=\begin{cases} a, \ if \ a>x \\ a\times x, \ if \ a\le x \end{cases}$
4	$y1=\begin{cases} 2-x, \ if \ x<2 \\ a+3, \ if \ x\ge 2 \end{cases}$	$y2=\begin{cases} a-1, \ if \ x<a \\ a\times x-1, \ if \ x\ge a \end{cases}$
5	$y1=\begin{cases} \lvert x\rvert, \ if \ x<0 \\ x-a, \ if \ x\ge 0 \end{cases}$	$y2=\begin{cases} a+x, \ if \ x \bmod 3=1 \\ 7, \ if \ x \bmod 3\ne 1 \end{cases}$
6	$y1=\begin{cases} x \bmod 4, \ if \ x>a \\ a, \ if \ x\le a \end{cases}$	$y2=\begin{cases} a\times x, \ if \ x/a>3 \\ x, \ if \ x/a\le 3 \end{cases}$
7	$y1=\begin{cases} 4-x, \ if \ \lvert x\rvert<3 \\ a+x, \ if \ \lvert x\rvert\ge 3 \end{cases}$	$y2=\begin{cases} 2, \ if \ x-\text{÷åò í î å} \\ 2a-x, \ if \ x-\text{í å÷åò í î å} \end{cases}$
8	$y1=\begin{cases} 4\times x, \ if \ x\le 4 \\ x-a, \ if \ x>4 \end{cases}$	$y2=\begin{cases} 7, \ if \ x-\text{í å÷åò í î å} \\ x/2+a, \ if \ x-\text{÷åò í î å} \end{cases}$
9	$y1=\begin{cases} a\times x, \ if \ x \bmod 3=2 \\ 9, \ if \ x \bmod 3\ne 2 \end{cases}$	$y2=\begin{cases} a-x, \ if \ a>x \\ a+2, \ if \ a\le x \end{cases}$
10	$y1=\begin{cases} a+\lvert x\rvert, \ if \ x>a \\ a-7, \ if \ x\le a \end{cases}$	$y2=\begin{cases} a\times 3, \ if \ a>3 \\ 11, \ if \ a\le 3 \end{cases}$
11	$y1=\begin{cases} 10+x, \ if \ x>1 \\ \lvert x\rvert+a, \ if \ x\le 1 \end{cases}$	$y2=\begin{cases} 2, \ if \ x>4 \\ x, \ if \ x\le 4 \end{cases}$

Таблица 4.1 (продолжение)

1	2	3
12	$y1=\begin{cases}15+x, \ if \ x>7\\ \lvert a\rvert-9, \ if \ x\le 7\end{cases}$	$y2=\begin{cases}3, \ if \ x>2\\ \lvert x\rvert-5, \ if \ x\le 2\end{cases}$
13	$y1=\begin{cases}3+x, \ if \ x=a\\ a-x, \ if \ x\ne a\end{cases}$	$y2=\begin{cases}\lvert a\rvert, \ if \ x>10\\ \lvert a\rvert-x, \ if \ x\le 10\end{cases}$
14	$y1=\begin{cases}2x+a, \ if \ x>2\\ 2x+1, \ if \ x\le 2\end{cases}$	$y2=\begin{cases}\lvert x\rvert+1, \ if \ x>0\\ a-1, \ if \ x\le 0\end{cases}$
15	$y1=\begin{cases}8+\lvert x\rvert, \ if \ x<1\\ \lvert a\rvert\times 2, \ if \ x\ge 1\end{cases}$	$y2=\begin{cases}3, \ if \ x=a\\ a+1, \ if \ x\ne a\end{cases}$
16	$y1=\begin{cases}4+x, \ if \ x\le 3\\ a\times x, \ if \ x>3\end{cases}$	$y2=\begin{cases}\lvert a\rvert-2, \ if \ x>a\\ x, \ if \ x\le a\end{cases}$
17	$y1=\begin{cases}a+\lvert x\rvert, \ if \ x<0\\ x-a, \ if \ x\ge 0\end{cases}$	$y2=\begin{cases}7, \ if \ x<3\\ a, \ if \ x\ge 3\end{cases}$
18	$y1=\begin{cases}7+x, \ if \ x<3\\ \lvert a\rvert+x, \ if \ x\ge 3\end{cases}$	$y2=\begin{cases}1, \ if \ x>5\\ a+x, \ if \ x\le 5\end{cases}$
19	$y1=\begin{cases}-5, \ if \ x>4\\ x-a, \ if \ x\le 4\end{cases}$	$y2=\begin{cases}\lvert a\rvert, \ if \ x>a\\ 9, \ if \ x\le a\end{cases}$
20	$y1=\begin{cases}2\times x, \ if \ x<5\\ \lvert a\rvert+x, \ if \ x\ge 5\end{cases}$	$y2=\begin{cases}3, \ if \ x<0\\ a+x, \ if \ x\ge 0\end{cases}$
21	$y1=\begin{cases}3, \ if \ x \bmod 3=1\\ x-a, \ if \ x \bmod 3\ne 1\end{cases}$	$y2=\begin{cases}a/x, \ if \ x\ne 0\\ 4, \ if \ x=0\end{cases}$
22	$y1=\begin{cases}\lvert a\rvert+\lvert x\rvert, \ if \ x<0\\ a\times x, \ if \ x\ge 0\end{cases}$	$y2=\begin{cases}3, \ if \ x=a\\ a-x, \ if \ x\ne a\end{cases}$
23	$y1=\begin{cases}2\times x, \ if \ \lvert x\rvert>4\\ 4+a, \ if \ \lvert x\rvert\le 4\end{cases}$	$y2=\begin{cases}9, \ if \ x=0\\ a/x, \ if \ x\ne a\end{cases}$
24	$y1=\begin{cases}x, \ if \ x \bmod 4\ne 2\\ a+x, \ if \ x \bmod 4=2\end{cases}$	$y2=\begin{cases}a-x, \ if \ x<a\\ a\times x, \ if \ x\ge a\end{cases}$
25	$y1=\begin{cases}12, \ if \ x<12\\ x+1, \ if \ x\ge 12\end{cases}$	$y2=\begin{cases}2, \ if \ x>2\\ a+x, \ if \ x\le 2\end{cases}$

Контрольные вопросы

1. В чем заключается преимущество использования библиотек динамической компоновки?

2. Чем отличается явное связывание от неявного связывания?

3. Этапы разработки DLL.

4. Из каких частей состоит библиотека динамической компоновки?

5. Структура сообщения передаваемого библиотеке.

6. В какую область адресного пространства процесса загружается DLL.

7. Адресное пространство процесса.

8. Причины вызова DLL-модуля.

Лабораторная работа № 5

Использование таймера

Цель работы: Получить навык написания программ, использующих таймер.

Краткие сведения из теории

Таймер является одним из мощных инструментов, предоставляемых операционной системой и позволяющих решать самые разнообразные задачи. Для работы с таймером в консольных приложениях используются функциии timeSetEvent и timeKillEvent. В оконных приложениях чаще используют функции SetTimer и KillTimer. Особенность таймера, создаваемого функцией SetTimer, заключается в том, что сообщение WM_TIMER, которое начинает посылать система приложению после выполнения функции SetTimer, приходит со всеми другими сообщениями наравне, на общих основаниях. Следовательно, интервал между двумя

приходами сообщения WM_TIMER может несколько варьироваться. В большинстве случаев это не существенно.

Если система посылает сообщение приложению, а предыдущее сообщение еще стоит в очереди, то система объединяет эти два сообщения. Таким образом, «вынужденный простой» не приводит к приходу на приложение подряд нескольких сообщений таймера.

Перечислим те задачи, которые можно решить с помощью таймера.

- Отслеживание времени: секундомер, часы и т.д. Нарушение периодичности здесь не имеет значения, так как по приходу сообщения время можно отследить, вызвав функцию получения системного времени.
- Таймер – один из способов осуществления многозадачности. Можно установить сразу несколько таймеров на разные функции, в результате периодически будет исполняться то одна, то другая функция.
- Периодический вывод на экран обновленной информации.
- Автосохранение – функция особенно полезная для редакторов.
- Задание темпа изменения каких-либо объектов на экране.
- Мультипликация – по приходе сообщения от таймера обновляется графическое содержимое экрана или окна, так что возникает эффект мультипликации.

Параметры функции setTimer:

1-й параметр – дескриптор окна, с которым ассоциируется таймер. Если этот параметр сделать равным NULL (0), то будет проигнорирован и второй параметр;

2-й параметр – определяет идентификатор таймера;

3-й параметр – определяет интервал посылки сообщения WM_TIMER;

4-й параметр – определяет адрес функции, на которую будет приходить сообщение WM_TIMER. Если параметр равен NULL, то сообщение будет приходить на функцию окна.

Если функция выполнилась успешно, то возвращаемым значением будет являться идентификатор таймера, который, естественно, будет совпадать со вторым параметром, если первый параметр будет отличным от NULL. В случае неудачи функция возвратит нуль.

Из сказанного следует, что функция может быть вызвана тремя способами: задан дескриптор окна, а четвертый параметр задается равным нулю; задан дескриптор окна, а четвертый параметр определяет функцию, на которую будет приходить сообщение WM_TIMER; дескриптор окна равен NULL, а четвертый параметр определяет функцию, на которую будет приходить сообщение WM_TIMER(идентификатор таймера в этом случае будет определяться по возвращаемому функцией значению).

Функция, на которую приходит сообщение WM_TIMER, имеет следующие параметры:

1-й параметр – дескриптор окна, с которым ассоциирован таймер;

2-й параметр – сообщение WM_TIMER;

3-й параметр – идентификатор таймера;

4-й параметр – время в миллисекундах, прошедшее с момента запуска Windows.

Функция KillTimer удаляет созданный параметр и имеет следующие параметры:

1-й параметр – дескриптор окна;

2-й параметр – идентификатор таймера.

Для работы с таймером необходимо определить константу

```
WM_TIMER      equu 113h
```

и прототипы внешних процедур:

```
EXTERN    SetTimer@16:NEAR
EXTERN    KillTimer@8:NEAR
```

Для компоновщика понадобится следующая библиотека:

```
includelib \masm32\lib\wmvcore.lib
```

Установка таймера, реализуется в оконной процедуре:

```
PUSH 0      ;Параметр = NULL
PUSH 1000   ;Интервал  1 с
PUSH 1      ;Идентификатор таймера
PUSH DWORD  PTR  [EBP+08H]
CALL    SetTimer@16
```

Удаление таймера

```
PUSH 1   ;Идентификатор таймера
PUSH DWORD  PTR  [EBP+08H]
CALL KillTimer@8
```

Обнаружение сообщения

```
CMP    DWORD PTR [EBP+0CH],WM_TIMER
```

Рассмотрим пример, в котором реализовано два таймера. Можно считать, что запускаются одновременно две задачи. Одна задача с периодичностью 0,5 сек. получает системное время и формирует строку для вывода (STRCOPY). Эта задача имеет свою собственную функцию, на которую приходит сообщение WM_TIMER. Вторая задача работает в рамках функции окна. Эта задача с периодичностью 1 сек. выводит время и дату в окно редактирования, расположенное на диалоговом окне. Таким образом, две задачи взаимодействуют друг с другом посредством глобальной переменной STRCOPY. Поскольку на функцию таймера приходит сообщение, в котором указан идентификатор таймера, мы можем на базе одной функции реализовать любое количество таймеров.

```
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
START:
```

```
;Получить дескриптор приложения.
   PUSH  0
   CALL  GetModuleHandleA@4
   MOV   [HINST], EAX
;Создать диалоговое окно.
   PUSH  0
   PUSH  OFFSET WNDPROC
   PUSH  0
   PUSH  OFFSET PA
   PUSH  [HINST]
   CALL  DialogBoxParamA@20
   CMP   EAX,-1
   JNE   KOL
;Здесь можно разместить сообщение об ошибке.
KOL:
   PUSH  0
   CALL  ExitProcess@4
WNDPROC PROC
      PUSH EBP
      MOV  EBP, ESP
      PUSH EBX
      PUSH ESI
      PUSH EDI
      CMP  DWORD PTR [EBP+0CH],WM_CLOSE
      JNE  L1
;Удалить таймер 1
      PUSH 1   ;Идентификатор таймера.
      PUSH DWORD  PTR  [EBP+08H]
      CALL KillTimer@8
;Удалить таймер 2
      PUSH 2   ;Идентификатор таймера.
      PUSH DWORD  PTR  [EBP+08H]
      CALL KillTimer@8
;Закрыть диалог
      PUSH 0
      PUSH DWORD PTR [EBP+08H]
      CALL EndDialog@8
      JMP  FINISH
L1:
      CMP  DWORD PTR [EBP+0CH],WM_INITDIALOG
      JNE  L2
```

```
;Установить таймер 1
      PUSH 0      ;Параметр = NULL
      PUSH 1000   ;Интервал  1 с
      PUSH 1      ;Идентификатор таймера
      PUSH DWORD  PTR  [EBP+08H]
      CALL SetTimer@16
;Установить таймер 2
      PUSH OFFSET TIMPROC  ;Параметр = NULL
      PUSH 500             ;Интервал 0.5 с
      PUSH 2               ;Идентификатор таймера
      PUSH DWORD  PTR  [EBP+08H]
      CALL SetTimer@16
      JMP  FINISH
L2:
      CMP  DWORD PTR [EBP+0CH],WM_TIMER
      JNE  FINISH
;Отправить строку в окно
      PUSH OFFSET  STRCOPY
      PUSH 0
      PUSH WM_SETTEXT
      PUSH 1  ;Идентификатор элемента.
      PUSH DWORD PTR [EBP+08H]
      CALL SendDlgItemMessageA@20
FINISH:
      POP  EDI
      POP  ESI
      POP  EBX
      POP  EBP
      MOV  EAX,0
      RET  16
WNDPROC ENDP
;------------------------------------
;Процедура таймера
;Расположение параметров в стеке.
;[EBP+014H]   LPARAM  - промежуток запуска Windows.
;[EBP+10H]    WAPARAM  - идентификатор таймера.
;[EBP+0CH]    WM_TIMER
;[EBP+8]      HWND
TIMPROC PROC
      PUSH EBP
      MOV  EBP,ESP
```

```
;Получить локальное время
      PUSH OFFSET DATA
      CALL GetLocalTime@4
;Получить  строку для вывода даты и времени
      MOVZX  EAX, DATA.sec
      PUSH EAX
      MOVZX  EAX, DATA.min
      PUSH EAX
      MOVZX  EAX,DATA.hour
      PUSH EAX
      MOVZX  EAX,DATA.year
      PUSH EAX
      MOVZX  EAX,DATA.month
      PUSH EAX
      MOVZX  EAX,DATA.day
      PUSH EAX
      PUSH OFFSET TIM
      PUSH OFFSET STRCOPY
      CALL  wsprintfA
;Восстановить стек
      ADD  ESP,32
      POP  EBP
      RET  16
TIMPROC ENDP
_TEXT   ENDS
      END START
```

Обратите внимание на весьма полезную функцию GetLocalTime. Информация, полученная с помощью этой функции, легко может быть использована для самых разных целей, в том числе и для вывода на экран. Аналогично, с помощью функции SetLocalTime, вы сможете установить текущее время. Для получения времени по Гринвичу используется функция GetSystemTime. Соответственно, с помощью SetSystemTime используется для установки времени в Гринвичском выражении.

Таблица 6.1 – Варианты заданий на лабораторную работу № 6

№	Задание на лабораторную работу

пп.	Т 1, сек.	Действие	Т 2, сек.	Действие
1	2	Открытие/закрытие CD-ROM	0,5	Счетчик сообщений WM_TIMER
2	3	Изменение текста в окне приложения	1	Отображение времени
3	1	Изменение размера окна	4	Инкремент даты
4	5	Создание/удаление файла	1,4	Изменение цвета окна
5	1,5	Изменение текста окна сообщения	3	Изменение цвета шрифта
6	0,8	Последовательный вывод текста	1,7	Счетчик сообщений WM_SIZE
7	1,7	Исчезновение и появление текста	2,5	Свернуть и развернуть окно
8	4,2	Отключение съемного устройства	0,9	Состояние съемного накопителя
9	2,5	Информация об изменении размера диска C:\	0,4	Счетчик сообщений WM_CHAR
10	4	Информация об изменении размера flash-диска	2,2	Список логических дисков
11	1,1	Вывод элементов арифметической прогрессии	5,1	Обновление содержимого файла с элементами
12	3,2	Вывод элементов геометрической прогрессии	1,2	Счетчик сообщений WM_RBUTTONDOWN
13	1,8	Счетчик сообщений WM_KEYDOWN	3,9	Центрирование курсора мыши
14	6	Отправка сообщения с помощью net send	0,3	Отображение координат положения курсора
15	3,5	Создание файла с именем равным текущему времени	1,5	Счетчик сообщений WM_KEYUP
16	2,3	Вывод элементов ряда Фибоначчи	4,4	Декремент 100 до 0 и сброс в 100
17	0,7	Вывод времени последнего изменения размера диска C:\	2,7	Обнуление счетчика WM_RBUTTONDOWN
18	4,5	Обнуление счетчика WM_MBUTTONDBLCLK	10	Создание MessageBox

Методика и порядок выполнения работы

1. Изучить принципы использования системного таймера.
2. Разработать оконное приложение, использующее таймер.
3. Защитить лабораторную работу преподавателю.

Контрольные вопросы

1. Что такое системный таймер?
2. Для каких целей используют системный таймер?
3. Как взаимодействует приложение с системным таймером?
4. Как организовать работу двух таймеров в одном приложении?
5. Особенности использования двух и более таймеров.

Лабораторная работа № 6

Создание и использование потоков

Цель работы: Получить навык написания программ, использующих потоки.

Краткие сведения из теории

Поток может быть создан при помощи функции CreateThread. Рассмотрим параметры этой функции.

1-й параметр – указатель на структуру атрибутов доступа. Имеет значение только для Windows NT. Обычно полагается NULL.

2-й параметр – размер стека потока. Если параметр равен нулю, то берется размер стека по умолчанию, равный размеру стека родительского потока.

3-й параметр – указатель на потоковую функцию, с вызова которой начинается исполнение потока.

4-й параметр – Параметр для потоковой функции.

5-й параметр – флаг, определяющий состояние потока. Если флаг равен 0, то выполнение потока начинается немедленно. Если значение флага потока равно CREATE_SUSPENDED (4H), то поток находится в состоянии ожидания и запускается по выполнению функции ResumeThread.

6-й параметр – Указатель на переменную, куда будет помещен дескриптор потока.

Выполнение потока начинается с потоковой функции. Окончание работы этой функции приводит к естественному окончанию работы потока. Поток также может закончить свою работу, выполнив функцию ExitThread с указанием кода выхода. Наконец, порождающий поток может закончить работу порожденного потока при помощи функции TerminateThread. В нижеприведенном примере запускаемый процесс не может сам закончить свою работу и прекращает свою работу вместе с приложением по команде TerminateThread. Надо сказать, что такое завершение является аварийным и не рекомендуется к обычному употреблению. Связано это с тем, что при таком завершении не выполняется никаких действий по освобождению занятых ресурсов (блоки памяти, открытые файлы и т. п.). Поэтому разработайте свое приложение так, чтобы поток завершался по выходу из потоковой процедуры.

Идеальной является ситуация, когда функция окна берет на себя только реакцию на события, происходящие с элементами, а всю трудоемкую работу (сложные вычисления, файловая обработка) должны взять на себя потоки. Поток может создавать новые потоки, так что в результате может возникнуть целое дерево.

Ниже представлена программа, использующая поток для вычисления и вывода в окно редактирования текущей даты и времени. Заметим в этой связи, что если бы такая обработка была реализована в оконной функции, вы бы сразу почувствовали разницу – окно почти бы перестало реагировать на внешнее воздействие.

```
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
START:
;Получить дескриптор приложения.
```

```
    PUSH  0
    CALL  GetModuleHandleA@4
    MOV   [HINST], EAX
;Создать диалоговое окно.
    PUSH  0
    PUSH  OFFSET WNDPROC
    PUSH  0
    PUSH  OFFSET PA
    PUSH  [HINST]
    CALL  DialogBoxParamA@20
    CMP   EAX,-1
    JNE   KOL
;Здесь можно разместить сообщение об ошибке.
KOL:
    PUSH  0
    CALL  ExitProcess@4
;Процедура окна.
WNDPROC PROC
      PUSH EBP
      MOV  EBP, ESP
      PUSH EBX
      PUSH ESI
      PUSH EDI
;Сообщение при закрытии окна.
      CMP  DWORD PTR [EBP+0CH],WM_CLOSE
      JNE  L1
L3:
;Здесь реакция на закрытие окна.
;Удалить поток.
      PUSH 0
      PUSH HTHR
      CALL TerminateThread@8
;Закрыть диалог.
      PUSH 0
      PUSH DWORD PTR [EBP+08H]
      CALL EndDialog@8
      JMP  FINISH
L1:
      CMP  DWORD PTR [EBP+0CH],WM_INITDIALOG
      JNE  L2
;Здесь начальная инициализация.
```

```
;Получить дескриптор окна редактирования.
      PUSH 1
      PUSH DWORD PTR [EBP+08H]
      CALL GetDlgItem@8
;Создать поток.
      PUSH OFFSET HTHR  ;Сюда дескриптор потока.
      PUSH 0
      PUSH EAX    ;Параметр.
      PUSH OFFSET GETTIME ;Адрес процедуры.
      PUSH 0
      PUSH 0
      CALL CreateThread@24
      JMP  FINISH
L2:
      CMP  DWORD PTR [EBP+0CH],WM_COMMAND
      JNE  FINISH
;Кнопка выхода?
      CMP  WORD PTR [EBP+10H],2
      JE   L3
FINISH:
      POP  EDI
      POP  ESI
      POP  EBX
      POP  EBP
      MOV  EAX,0
      RET  16
WNDPROC ENDP
;------------------------------------
;Потоковая функция.
; [EBP+8]  параметр=дескриптор окна редактирования
GETTIME PROC
      PUSH EBP
      MOV  EBP,ESP
L0:
;Задержка в 1 секунду.
      PUSH 1000
      CALL Sleep@4
;Получить локальное время.
      PUSH OFFSET DATA
      CALL GetLocalTime@4
;Получить строку для вывода даты и времени.
```

```
        MOVZX  EAX,DATA.sec
        PUSH EAX
        MOVZX  EAX,DATA.min
        PUSH EAX
        MOVZX  EAX,DATA.hour
        PUSH EAX
        MOVZX  EAX,DATA.year
        PUSH EAX
        MOVZX  EAX,DATA.month
        PUSH EAX
        MOVZX  EAX,DATA.day
        PUSH EAX
        PUSH OFFSET TIM
        PUSH OFFSET STRCOPY
        CALL  wsprintfA
;Отправить строку в окно редактирования.
        PUSH OFFSET STRCOPY
        PUSH 0
        PUSH WM_SETTEXT
        PUSH DWORD PTR [EBP+08H]
        CALL  SendMessageA@16
        JMP  L0 ;Бесконечный цикл.
        POP  EBP
        RET  4
GETTIME ENDP
_TEXT   ENDS
        END START
```

Возьмите на вооружение весьма полезную функцию Sleep. Эта функция особенно часто используется именно в потоках, с тем чтобы высвободить процессорное время.

Методика и порядок выполнения работы

1. Изучить принципы создания и использования потоков.

2. Разработать оконное приложение, использующее потоки. Задание на лабораторную работу взять из таблицы 6.1.

3. Защитить лабораторную работу преподавателю.

Контрольные вопросы

1. Что такое поток?
2. В чем преимущество использования потоков?
3. Как создаются потоки?
4. Как правильно завершать работу потоков?
5. Как создается дерево потоков?

Лабораторная работа № 7

Многопотоковое программирование

Цель работы: Получить навык создания многопотоковых приложений.

Краткие сведения из теории

Семафор представляет собой глобальный объект, позволяющий синхронизировать работу двух или нескольких процессов или потоков. Для программиста семафор – это просто счетчик (хотя манипулировать им можно только при помощи специальных функций). Если счетчик равен N, это означает, что к ресурсу имеют доступ N процессов. Рассмотрим функции для работы с семафорами.

CreateSemaphor – создает глобальный объект-семафор. Возвращает дескриптор семафора. Параметры функции:

1-й параметр. Указатель на структуру, определяющую атрибуты доступа. Может иметь значение для Windows NT. Обычно данный параметр полагается равным NULL.

2-й параметр. Начальное значение счетчика семафора. Определяет, сколько задач имеют доступ к ресурсу вначале.

3-й параметр. Количество задач, которые имеют одновременный доступ к ресурсу.

4-й параметр. Указатель на строку, содержащую имя семафора.

OpenSemaphor – открыть уже созданный семафор. Возвращает дескриптор семафора. Данную функцию используют не так часто. Обычно создают семафор и присваивают его дескриптор глобальной переменной, а потом используют этот дескриптор в порождаемых потоках. Параметры функции:

1-й параметр. Определяет желаемый уровень доступа к семафору. Возможные значения:

SEMAPHORE_MODIFY_STATE (2h) – разрешить использование функции ReleaseSemaphore,

SYNCHRONIZE (100000h) – разрешить использование любой функции ожидания, только для Windows NT,

SEMAPHORE_ALL_ACCESS (0F0000h + SYNCHRONIZE + 3h) – специфицирует все возможные флаги доступа к семафору.

WaitForSingleObject – ожидать открытия семафора. При успешном завершении, т.е. открытии доступа к объекту, функция возвращает 0. Значение 102h говорит о том, что заданный период ожидания закончился. Параметры функции:

1-й параметр. Дескриптор семафора.

2-й параметр. Время ожидания в миллисекундах. Если параметр равен INFINITE (0FFFFFFFFh), то время ожидания не ограничено.

ReleaseSemaphor – освоб дить семафор и тем самым открыть доступ к ресурсу другим процессам. Параметры функции:

1-й параметр. Дескриптор семафора.

2-й параметр. Определяет, какое значение должно быть добавлено к счетчику семафора. Чаще всего этот параметр равен единице.

3-й параметр. Указатель на переменную, куда должно быть помещено предыдущее значение счетчика.

Рассмотрим алгоритм работы с семафором. Сначала при помощи функции CreateSemaphor создается семафор и его дескриптор присваивается глобальной переменной. Перед попыткой обращения к ресурсам, доступ к которым необходимо ограничить, поток должен вызвать функцию WaitForSingleObject. При открытии доступа функция возвращает 0. По окончании работы с ресурсом следует вызвать функцию ReleaseSemaphor. Тем самым увеличивается счетчик доступа на 1. С помощью семафора можно регулировать количество потоков, которые одновременно могут иметь доступ к ресурсу. Максимальное значение счетчика как раз и определяет, сколько потоков могут получить доступ к ресурсу одновременно. Но обычно максимальное значение полагают равным 1.

Событие является объектом, очень похожим на семафор, но в несколько видоизмененном виде. Рассмотрим функции для работы с событиями.

CreateEvent - сздает объект-событие. Параметры функции:

1-й параметр. Имеет тот же смысл, что и первый параметр функции CreateSemaphor. Обычно полагается равным NULL.

2-параметр. Если параметр не равен нулю, то событие может быть сброшено при помощи функции ResetEvent. Иначе событие сбрасывается при доступе к нему какого-либо процесса.

3-й параметр. Если параметр равен 0, то событие инициализируется как сброшенное, в противном случае сразу же подается сигнал о наступлении соответствующей ситуации.

4-й параметр. Указатель на строку, которая содержит имя события.

Ожидание события осуществляется, как и в случае с семафором, функцией WaitForSingleObject.

Функция OpenEvent аналогична функции OpenSemaphor, и на ней мы останавливаться не будем.

SetEvent – подать сигнал о наступлении события. Параметры функции:

1-й параметр. Дескриптор события.

Критическая секция – это фрагмент программы, защищенный от одновременного выполнения несколькими потоками. Критическую секцию в данный момент может выполнять только один поток. Рассмотрим функции для работы с критической секцией.

InitializeCriticaiSection – данная функция создает объект под названием критическая секция. Параметры функции:

1-й параметр. Указатель на структуру, указанную ниже. Поля данной структуры используются только внутренними процедурами, и смысл безразличен.

```
CRITICAL_SECTION  STRUCT
        DebugInfo       DWORD ?
        LockCount       LONG ?
        RecursionCount  LONG ?
        OwningThread    HANDLE ?
        LockSemaphore   HANDLE ?
        SpinCount       DWORD ?
CRITICAL_SECTION  ENDS
```

EnterCriticalSection – войти в критическую секцию. После выполнения этой функции данный поток становится владельцем данной секции. Следующий поток, вызвав данную функцию, будет находиться в состоя-

нии ожидания. Параметр функции такой же, что и в предыдущей функции.

LeaveCriticalSection – покинуть критическую секцию. После этого второй поток, который был остановлен функцией EnterCriticalSection, станет владельцем критической секции. Параметр функции LeaveCriticalSection такой же, как и у предыдущих функций.

DeleteCriticalSection - удалить объект «критическая секция». Параметр аналогичен предыдущим.

Программно можно определить несколько объектов критической секции, с которыми будут работать несколько потоков. Мы не зря, говоря о критических секциях, упоминаем только потоки. Разные процессы не могут использовать данный объект синхронизации.

Мьютексы (Mutex) это объекты ядра, которые создаются функцией CreateMutex(). Мьютекс бывает в двух состояниях - занятом и свободном. Мьютексом хорошо защищать единичный ресурс от одновременного обращения к нему разными потоками.

Методика и порядок выполнения работы

1. Изучить принципы создания и использования потоков.

2. Написать программу, которая создаёт несколько потоков и синхронизирует их работу с использованием инструментов Critical Section, mutex и semaphore. Для управления работой потоков использовать только функции синхронизации.

3. Защитить лабораторную работу преподавателю.

Таблица 8.1 – Задания на лабораторную работу № 8.

№	Индивидуальное задание
1	Поток считает сумму ряда по 100 элементов, после чего он переходит в режим ожидания, пока в основном потоке пользователь не нажмет

	любую клавишу. На экран выводить текущее значение суммы ряда. Использовать mutex.
2	Создать 2 потока, которые по очереди считают значение определенного интеграла на заданном интервале, причем каждый поток считает только значение только на своём интервале. Оба потока используют одни и те же глобальные переменные для расчётов. Текущее значение интеграла и номер активного потока выводиться на экран. Использовать Critical section.
3	Создать 2 потока. Первый поток уменьшает значение какой либо глобальной переменной на фиксированное значение, второй – увеличивает то же значение. При нажатии на клавиши поток блокируется/активизируется. Значение величины и состояние каждого потока выводится на экран. Использовать mutex.
4	Создать 4 потока, которые увеличивают значение целой глобальной переменной. При достижении определенного значения переменная обнуляется. Пользователь с клавиатуры задаёт число одновременно работающих потоков. Текущее значение переменной выводится на экран. Использовать semaphore.
5	Создать 3 потока, которые через случайное время (50 – 1500 мс) меняют значение глобальной переменной на случайную величину (-10 … 10). После чего блокируют изменение этой переменной на это же время. Использовать функцию sleep и critical section. Отображать на экране значение этой переменной, и номер потока, изменивший её значение в последний раз.
6	Создать 4 потока, которые через равные интервалы времени (1 сек) меняют в строке местами 2 произвольных символа. Пользователь с клавиатуры задаёт количество одновременно работающих потоков. На экране отображать текущее состояние строки.
7	Создать 3 потока, которые по очереди рассчитывают значения функции на заданном диапазоне с заданным шагом. Каждый поток рассчитывает только по одному значению. На экран выводить текущее значение аргумента и функции, номер потока, рассчитавшего последнее значение. Использовать mutex.
8	Создать 2 потока. При нажатии на клавиши клавиатуры активизируется первый поток на какое-то случайное время (300-2000 мс). Каждые 200 мс он разрешает/запрещает расчет суммы числового ряда, которая считается вторым потоком. На экран выводить текущее значение суммы ряда. Использовать mutex.
9	Создать 2 потока. Первый через произвольное время (200-1000 мс) удаляет из начала строки символ (если строка не пуста) и блокирует

	изменение строки на 200 мс. Второй поток – добавляет символ в конец строки через 300-1200 мс, и блокирует изменение строки на 300мс. На экране отображать текущую строку, и номер потока, последним менявший строку. Использовать critical section.
10	Создать 2 потока, которые меняют значения 2х элементов массива в произвольной позиции по следующим правилам. Первый поток к значению большего элемента прибавляет 1, а от значения меньшего – отнимает 1. Второй поток – к меньшему прибавляет 1, от большего элемента – отнимает 1. Пользователь при нажатии клавиши моет блокировать/разрешать работу каждого потока. На экран выводить текущие значения элементов массива.
11	Создать 2 потока. Первый поток удаляет первый символ строки каждые 500 мс. Второй поток добавляет символ в конец строки через случайные промежутки времени (100 – 1000 мс). При нажатии на клавиши блокируется/разрешается работа каждого потока в отдельности. Для управления работой потоков использовать critical section.
12	Создать 3 потока. Первый поток добавляет символ «1» в конец строки через 250 – 2000 мс. Второй поток добавляет символ «2» в начало строки через 250 – 2000 мс. Третий поток удаляет символ из середины строки через каждые 250 мс. Пользователь с клавиатуры задает количество одновременно работающих потоков (0-3). Использовать semaphore.
13	Создать 2 потока. Первый поток циклически сдвигает символы текста вправо через случайные промежутки времени (100 – 2000 мс), второй поток – сдвигает влево. После сдвига изменение текста блокируется на время 500 мс. Использовать critical section и функцию sleep.
14	Создать 3 потока. Каждый поток считает сумму 1000 значений функции y=sin(x) с шагом 0.0001. Для вычислений использовать глобальные переменные. Для синхронизации вычислений использовать critical section.
15	Создать 3 потока, каждый из которых через произвольное время (100 – 3000 мс) удаляет из начала строки символ и добавляет в конец строки свой номер. После чего блокирует строку от изменений на 250 мс. Использовать critical section
16	Создать 2 потока. Первый поток генерирует через 100-500 мс случайное число. Второй поток считает среднее арифметическое этих случайных чисел. Значения выводятся на экран. Использовать mutex.
17	Создать 3 потока. Первый поток циклически сдвигает текст влево, второй – сдвигает циклически вправо, третий – добавляет/удаляет символ в/из середины строки. Потоки выполняют свои действия через

	100 – 2000 мс, после чего блокируют изменение строки на 200 мс. Если строка блокирована, то время до выполнения потоком следующей итерации удваивается. Использовать функцию sleep и critical section.
18	Создать 2 потока. Первый поток считает значение функции y=cos(x), начиная от 0 с шагом 0.001. Второй поток считает сумму значений этой функции. Использовать mutex.
19	Создать 2 потока, которые производят циклический сдвиг цифр числа через каждые 200 мс. Пользователь, нажимая цифровые на клавиши, задаёт количество итераций, выполняемых потоками. Использовать semaphore.
20	Создать 2 потока. Первый поток считает сумму положительных чисел, введенных пользователем с клавиатуры, второй поток – сумму отрицательных. Текущие значения выводить на экран. Использовать mutex.

Контрольные вопросы

1. Чем отличаются процессы и потоки?
2. Средства синхронизации потоков в WinAPI.
3. Преимущества многопоточных приложений.
4. В каких случаях необходимо использовать критические секции?
5. Отличие семафора от события.
6. Алгоритм использования семафора.
7. Для чего необходима синхронизация потоков?

Рекомендуемая литература

Основная литература

1. Новиков Ю.В. Основы микропроцессорной техники. — Электрон. текст. дан.— М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — Режим доступа : http ://www.iprbookshop.ru/52207. — ЭБС «IPRbooks», по паролю.

2. Водовозов А.М. Микроконтроллеры для систем автоматики : Учеб. пособие. — Электрон. текст. дан. — М. : Инфра-Инженерия, 2016. — Режим доступа : http ://www.iprbookshop.ru/51727.— ЭБС «IPRbooks», по паролю

Дополнительная литература

1. Гуров В.В. Архитектура микропроцессоров [Электронный ресурс]/ Гуров В.В.— Электрон. текстовые данные.— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 115 с.— Режим доступа: http://www.iprbookshop.ru/56313.html.— ЭБС «IPRbooks»

2. Аблязов Р.З. Программирование на ассемблере на платформе x86-64 [Электронный ресурс]/ Аблязов Р.З.— Электрон. текстовые данные.— Саратов: Профобразование, 2017.— 304 с.— Режим доступа: http://www.iprbookshop.ru/63951.html.— ЭБС «IPRbooks»

Приложение А

```
.386P
.MODEL FLAT, stdcall

; сообщение приходит при закрытии окна
WM_DESTROY        equ 2
; сообщение приходит при создании окна
WM_CREATE         equ 1
; сообщение при щелчке левой кнопкой мыши в области окна
WM_LBUTTONDOWN    equ 201h
; сообщение при щелчке правой кнопкой мыши в области окна
WM_RBUTTONDOWN    equ 204h

; свойства окна
CS_VREDRAW        equ 1h
CS_HREDRAW        equ 2h
CS_GLOBALCLASS    equ 4000h
WS_OVERLAPPEDWINDOW    equ 000CF0000H
style equ CS_HREDRAW+CS_VREDRAW+CS_GLOBALCLASS

; идентификатор стандартной иконки
IDI_APPLICATION   equ 32512
; идентификатор курсора
IDC_CROSS         equ 32515

; режим показа окна – нормальный
SW_SHOWNORMAL     equ 1

; прототипы внешних процедур
EXTERN   MessageBoxA@16:  NEAR
EXTERN   CreateWindowExA@48:NEAR
EXTERN   DefWindowProcA@16:NEAR
EXTERN   DispatchMessageA@4:NEAR
EXTERN   ExitProcess@4:NEAR
EXTERN   GetMessageA@16:NEAR
EXTERN   GetModuleHandleA@4:NEAR
EXTERN   LoadCursorA@8:NEAR
EXTERN   LoadIconA@8:NEAR
```

```
EXTERN   PostQuitMessage@4:NEAR
EXTERN   RegisterClassA@4:NEAR
EXTERN   ShowWindow@8:NEAR
EXTERN   TranslateMessage@4:NEAR
EXTERN   UpdateWindow@4:NEAR

; директивы компоновщику для подключения библиотек
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib
;-----------------------------------------------
; структура сообщения
MSGSTRUCT  STRUC
   MSHWND        DD   ? ; идентификатор окна,
                        ; получающего сообщение
   MSMESSAGE     DD   ? ; идентификатор сообщения
   MSWPARAM      DD   ? ; доп. информация о сообщении
   MSLPARAM      DD   ? ; доп. информация о сообщении
   MSTIME        DD   ? ; время посылки сообщения
   MSPT          DD   ? ; положение курсора, во время посылки
                        ; сообщения
MSGSTRUCT  ENDS
;---------
WNDCLASS  STRUC
   CLSSTYLE      DD   ? ; стиль окна
   CLWNDPROC     DD   ? ; указатель на процедуру окна
   CLSCSEXTRA    DD   ? ; информация о доп. байтах для
                        ; данной структуры
   CLWNDEXTRA    DD   ? ; информация о доп. байтах для окна
   CLSHINSTANCE  DD   ? ; дескриптор приложения
   CLSHICON      DD   ? ; идентификатор иконы окна
   CLSHCURSOR    DD   ? ; идентификатор курсора окна
   CLBKGROUND    DD   ? ; идентификатор кисти окна
   CLMENUNAME    DD   ? ; имя-идентификатор меню
   CLNAME        DD   ? ; специфицирует имя класса окон
WNDCLASS  ENDS

; сегмент данных
_DATA SEGMENT  DWORD  PUBLIC  USE32 'DATA'
   NEWHWND    DD 0
   MSG      MSGSTRUCT  <?>
   WC       WNDCLASS   <?>
```

```
    HINST      DD 0 ; здесь хранится дескриптор приложения
    TITLENAME  DB 'Простой пример 32-битного приложения',0
    CLASSNAME  DB 'CLASS32',0
    CAP        DB 'Сообщение',0
    MES1       DB 'Вы нажали левую кнопку мыши',0
    MES2       DB 'Выход из программы. Пока!',0
_DATA ENDS

; сегмент кода
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
START:
; получить дескриптор приложения
    PUSH    0
    CALL    GetModuleHandleA@4
    MOV     [HINST], EAX

REG_CLASS:
; заполнить структуру окна стиль
    MOV [WC.CLSSTYLE], style
    MOV [WC.CLWNDPROC], OFFSET WNDPROC
    MOV [WC.CLSCSEXTRA], 0
    MOV [WC.CLWNDEXTRA], 0
    MOV EAX, [HINST]
    MOV [WC.CLSHINSTANCE], EAX

;--------- иконка окна
    PUSH IDI_APPLICATION
    PUSH 0
    CALL LoadIconA@8
    MOV  [WC.CLSHICON], EAX

;---------- курсор окна
    PUSH IDC_CROSS
    PUSH 0
    CALL LoadCursorA@8
    MOV  [WC.CLSHCURSOR], EAX
;---------регистрируем класс
    MOV  [WC.CLBKGROUND], 17 ; цвет окна
    MOV DWORD PTR [WC.CLMENUNAME], 0
    MOV DWORD PTR [WC.CLNAME], OFFSET CLASSNAME
    PUSH OFFSET WC
```

```
    CALL RegisterClassA@4

; создать окно зарегистрированного класса
    PUSH 0
    PUSH [HINST]
    PUSH 0
    PUSH 0
    PUSH 400 ; DY – высота окна
    PUSH 400 ; DX – ширина окна
    PUSH 100 ; Y – координата левого верхнего угла
    PUSH 100 ; X – координата левого верхнего угла
    PUSH WS_OVERLAPPEDWINDOW
    PUSH OFFSET TITLENAME  ; имя окна
    PUSH OFFSET CLASSNAME  ; имя класса
    PUSH 0
    CALL CreateWindowExA@48

; проверка на ошибку
    CMP EAX, 0
    JZ  _ERR
    MOV [NEWHWND], EAX ; дескриптор окна
; ------------------------------------------------
    PUSH SW_SHOWNORMAL
    PUSH [NEWHWND]
    CALL ShowWindow@8; показать созданное окно
; ------------------------------------------------
    PUSH [NEWHWND]
    CALL UpdateWindow@4   ; команда перерисовать видимую
                          ; часть окна, сообщение WM_PAINT
; петля обработки сообщений
MSG_LOOP:
    PUSH 0
    PUSH 0
    PUSH 0
    PUSH OFFSET MSG
    CALL GetMessageA@16
    CMP EAX, 0
    JE  END_LOOP
    PUSH OFFSET MSG
    CALL TranslateMessage@4
    PUSH OFFSET MSG
```

```
    CALL DispatchMessageA@4
    JMP MSG_LOOP

END_LOOP:
; выход из программы (закрыть процесс)
    PUSH [MSG.MSWPARAM]
    CALL ExitProcess@4

_ERR:
    JMP END_LOOP
; ----------------------------------------------
; процедура окна
; расположение параметров в стеке
; [EBP+014H] LPARAM
; [EBP+10H] WPARAM
; [EBP+0CH] MES
; [EBP+8] HWND
WNDPROC  PROC
    PUSH EBP
    MOV EBP, ESP
    PUSH EBX
    PUSH ESI
    PUSH EDI
CMP DWORD PTR [EBP+0CH], WM_DESTROY
    JE  WMDESTROY
CMP DWORD PTR [EBP+0CH], WM_CREATE
    JE  WMCREATE
CMP DWORD PTR [EBP+0CH] ,WM_LBUTTONDOWN    ;левая кнопка
    JE  LBUTTON
CMP DWORD PTR [EBP+0CH] ,WM_RBUTTONDOWN    ;правая кнопка
    JE  RBUTTON
    JMP DEFWNDPROC

; нажатие правой кнопки приводит к закрытию окна
RBUTTON:
    JMP WMDESTROY

; нажатие левой кнопки мыши
LBUTTON:
; выводим сообщение
    PUSH 0 ; MB_OK
```

```
    PUSH OFFSET CAP
    PUSH OFFSET MES1
    PUSH DWORD PTR [EBP+08H]
    CALL MessageBoxA@16
    MOV EAX, 0
    JMP FINISH

WMCREATE:
    MOV EAX, 0
    JMP FINISH

DEFWNDPROC:
    PUSH DWORD PTR [EBP+14H]
    PUSH DWORD PTR [EBP+10H]
    PUSH DWORD PTR [EBP+0CH]
    PUSH DWORD PTR [EBP+08H]
    CALL DefWindowProcA@16
    JMP FINISH

WMDESTROY:
    PUSH 0        ; MB_OK
    PUSH OFFSET CAP
    PUSH OFFSET MES2
    PUSH DWORD PTR [EBP+08H] ; дескриптор окна
    CALL MessageBoxA@16
    PUSH 0
    CALL PostQuitMessage@4  ; сообщение WM_QUIT
    MOV EAX, 0

FINISH:
    POP EDI
    POP ESI
    POP EBX
    POP EBP
    RET 16
WNDPROC ENDP
_TEXT ENDS
END START
```

Приложение Б

Индивидуальное задание на лабораторную работу №1

№	Обрабатываемые сообщения	Свойства окна	Режим показа окна	Цвет фона
1	WM_DESTROY WM_CREATE WM_MOVE WM_CHAR(ввод «г»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_HIDE	COLOR_SCROLLBAR
2	WM_DESTROY WM_CREATE WM_SIZE WM_CHAR(ввод «b»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWNORMAL	COLOR_BACKGROUND
3	WM_DESTROY WM_CREATE WM_MOUSEMOVE WM_CHAR(ввод «ю»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWMINIMIZED	COLOR_ACTIVECAPTION
4	WM_DESTROY WM_CREATE WM_LBUTTONDOWN WM_CHAR(ввод «d»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWMAXIMIZED	COLOR_INACTIVECAPTION
5	WM_DESTROY WM_CREATE WM_LBUTTONUP WM_CHAR(ввод «я»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_MAXIMIZE	COLOR_MENU
6	WM_DESTROY WM_CREATE WM_RBUTTONDOWN WM_CHAR(ввод «f»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWNOACTIVATE	COLOR_WINDOW

7	WM_DESTROY WM_CREATE WM_RBUTTONUP WM_CHAR(ввод «g»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOW	COLOR_WINDOWFRAME
8	WM_DESTROY WM_CREATE WM_MBUTTONDOWN WM_CHAR(ввод «h»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_MINIMIZE	COLOR_MENUTEXT
9	WM_DESTROY WM_CREATE WM_MBUTTONUP WM_CHAR(ввод «I»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWMINNOACTIVE	COLOR_WINDOWTEXT
10	WM_DESTROY WM_CREATE WM_LBUTTONDBLCLK WM_CHAR(ввод «J»)	CS_VREDRAW CS_HREDRAW CS_DBLCLKS	SW_SHOWNA	COLOR_CAPTIONTEXT
11	WM_DESTROY WM_CREATE WM_RBUTTONDBLCLK WM_CHAR(ввод «Д»)	CS_VREDRAW CS_HREDRAW CS_DBLCLKS	SW_RESTORE	COLOR_ACTIVEBORDER
12	WM_DESTROY WM_CREATE WM_MBUTTONDBLCLK WM_CHAR(ввод «L»)	CS_VREDRAW CS_HREDRAW CS_DBLCLKS	SW_SHOWDEFAULT	COLOR_INACTIVEBORDER
13	WM_DESTROY WM_CREATE WM_CHAR(ввод «Ф») SIZE_MAXHIDE	CS_VREDRAW CS_HREDRAW CS_NOCLOSE	SW_FORCEMINIMIZE	COLOR_APPWORKSPACE

14	WM_DESTROY WM_CREATE WM_CHAR(ввод «N») SIZE_MAXSHOW	CS_VREDRAW CS_HREDRAW CS_NOCLOSE	SW_HIDE	COLOR_HIGHLIGHT
15	WM_DESTROY WM_CREATE WM_CHAR(ввод «ц») SIZE_MAXIMIZED	CS_VREDRAW CS_HREDRAW CS_NOCLOSE	SW_SHOWNORMAL	COLOR_HIGHLIGHTTEXT
16	WM_DESTROY WM_CREATE WM_CHAR(ввод «Ш») SIZE_MINIMIZED	CS_VREDRAW CS_HREDRAW CS_NOCLOSE	SW_SHOWMINIMIZED	COLOR_BTNFACE
17	WM_DESTROY WM_CREATE WM_CHAR(ввод «Q») WM_MENUSELECT	CS_VREDRAW CS_HREDRAW CS_NOCLOSE	SW_SHOWMAXIMIZED	COLOR_BTNSHADOW
18	WM_DESTROY WM_CREATE WM_CHAR(ввод «S») WM_SETCURSOR	CS_VREDRAW CS_HREDRAW CS_NOCLOSE	SW_MAXIMIZE	COLOR_GRAYTEXT
19	WM_DESTROY WM_CREATE WM_CHAR(ввод «п») WM_SHOWWINDOW	CS_VREDRAW CS_HREDRAW CS_NOCLOSE	SW_SHOWNOACTIVATE	COLOR_BTNTEXT
20	WM_DESTROY WM_CREATE WM_CHAR(ввод «U») WM_KEYUP	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOW	COLOR_INACTIVECAPTIONTEXT

21	WM_DESTROY WM_CREATE WM_CHAR(ввод «W») WM_KEYDOWN	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_MINIMIZE	COLOR_BTNHIGHLIGHT
22	WM_DESTROY WM_CREATE WM_CHAR(ввод «X») WM_SYSKEYUP	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWMINNOACTIVE	COLOR_3DDKSHADOW
23	WM_DESTROY WM_CREATE WM_CHAR(ввод «Y») WM_SYSKEYDOWN	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWNA	COLOR_3DLIGHT
24	WM_DESTROY WM_CREATE WM_CHAR(ввод «Z») WM_SYSDEADCHAR	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_RESTORE	COLOR_INFOTEXT
25	WM_DESTROY WM_CREATE WM_CHAR(ввод «q») WM_SYSCHAR	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWDEFAULT	COLOR_INFOBK

Приложение В

Таблица В.1 – Сообщения операционной системы Windows

Сообщение системы	Назначение
WM_ACTIVATE	Посылается функции окна перед активизацией и деактивизацией этого окна.
WM_ACTIVATEAPP	Посылается функции окна перед активизацией окна другого приложения.
WM_CHAR	Сообщение, возникающее при трансляции сообщения WM_KEYDOWN функцией TranslateMessage.
WM_CLOSE	Сообщение, приходящее на процедуру окна при его закрытии. Приходит до WM_DESTROY. Дальнейшее выполнение DefWindowProc, EndDialog или WindowsDestroy и вызывает появление сообщения WM_DESTROY.
WM_COMMAND	Сообщение, приходящее на функцию окна, при наступлении события с управляющим элементом, пунктом меню, а также от акселератора.
WM_CREATE	Первое сообщение, приходящее на функцию окна при его создании. Приходит один раз.
WM_DEADCHAR	Сообщение, возникающее при трансляции сообщения WM_KEYUP функцией TranslateMessage.
WM_DESTROY	Сообщение, приходящее на функцию окна при его уничтожении.
WM_GETTEXT	Посылается окну для получения текстовой строки, ассоциированной с данным окном (строка редактирования, заголовок окна и т.д.).
WM_HOTKEY	Генерируется при нажатии горячей клавиши.
WM_INITDIALOG	Сообщение, приходящее на функцию диалогового окна вместо сообщения WM_CREATE.
WM_KEYDOWN	Сообщение, генерируемое при нажатии клавиши клавиатуры и посылаемое окну, имеющему фокус ввода.
WM_KEYUP	Сообщение, генерируемое при отпускании клавиши клавиатуры и посылаемое окну, имеющему фокус ввода.
WM_LBUTTONDOWN	Сообщение генерируется при нажатии левой кнопки мыши.
WM_MENUSELECT	Посылается окну, содержащему меню, при выборе пункта меню.
WM_PAINT	Сообщение посылается окну перед его перерисовкой.
WM_QUIT	Сообщение, приходящее приложению (не окну) при выполнении функции PostQuitMessage. При получе-

Сообщение системы	Назначение
	нии этого сообщения происходит выход из цикла ожидания и, как следствие, выход из программы.
WM_RBUTTONDOWN	Сообщение генерируется при нажатии правой кнопки мыши.
WM_SETFOCUS	Сообщение, посылаемое окну, после того, как оно получило фокус.
WM_SETICON	Приложение посылает окну данное сообщение, чтобы ассоциировать с ним новую иконку (значок).
WM_SETTEXT	Сообщение, используемое приложением для посылки текстовой строки окну и интерпретируемое в зависимости от типа окна (обычное окно - заголовок, кнопка — надпись на кнопке, окно редактирования - содержимое этого окна и т.д.).
WM_SIZE	Посылается функции окна после изменения его размера.
WM_SYSCHAR	Сообщение, возникающее при трансляции сообщения WM_SYSKEYDOWN функцией TranslateMessage.
WM_SYSCOMMAND	Генерируется при выборе пунктов системного меню или меню окна.
WM_SYSDEADCHAR	Сообщение, возникающее при трансляции сообщения WM_SYSKEYUP функцией TranslateMessage.
WM_SYSKEYDOWN	Сообщение аналогично WM_KEYDOWN, но генерируется, когда нажата и удерживается еще и клавиша Alt.
WM_SYSKEYUP	Сообщение аналогично WM_SYSKEYDOWN, но генерируется при отпускании клавиши.
WM_TIMER	Сообщение, приходящее на функцию окна или специально определенную таймерную процедуру после определения интервала таймера при помощи функции SetTimer.
WM_VKEYTOITEM	Сообщение окну приложения, когда нажимается какая-либо клавиша при наличии фокуса на данном списке. Список должен иметь свойство LBS_WANTKEYBOARDINPUT.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

НЕВИННОМЫССКИЙ ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)

Методические указания для выполнения практических работ

по дисциплине

«Управляющие микропроцессорные комплексы»

Методические указания квыполнению практических работ

Направление подготовки 15.04.04

«Автоматизация технологических процессов и производств»

Направленность (профиль) «Информационно-управляющие системы»

Невинномысск 2024

Методические указания предназначены для студентов очно-заочной формы направления подготовки 15.04.04 Автоматизация технологических процессов и производств и других технических специальностей. Они содержат основы теории, порядок проведения практических работ и обработки экспериментальных данных, перечень контрольных вопросов для самоподготовки и список рекомендуемой литературы. Работы подобраны и расположены в соответствии с методикой изучения дисциплины «Управляющие микропроцессорные комплексы». Объем и последовательность выполнения работ определяются преподавателем в зависимости от количества часов, предусмотренных учебным планом дисциплины, как для очной, так и для заочной форм обучения.

Методические указания разработаны в соответствии с требованиями Федерального Государственного образовательного стандарта в части содержания и уровня подготовки выпускников по специальности 15.04.04 Автоматизация технологических процессов и производств

Код	Формулировка
ПК-3	Способность: составлять описание принципов действия и конструкции устройств, проектируемых технических средств и систем автоматизации, управления, контроля, диагностики и испытаний технологических процессов и производств общепромышленного и специального назначения для различных отраслей национального хозяйства; проектировать их архитектурно-программные комплексы

Составитель: канд. техн. наук А.А. Евдокимов

Ответственный редактор: канд. техн. наук Д.В. Болдырев

Содержание

Практическая работа № 1 4
Практическая работа № 2 24
Практическая работа № 3 32
Практическая работа № 4 39
Литература 75

Практическая работа № 1

Основы программирования на языке ассемблера процессора 8086

Цель работы: ознакомиться с основами программирования на языке ассемблера. Научиться создавать, компилировать и компоновать программы на языке ассемблера.

Краткие сведения из теории

31 16	15 0	
	AH AX AL	EAX
	BH BX BL	EBX
	CH CX CL	ECX
	DH DX DL	EDX
	SI	ESI
	DI	EDI
	BP	EBP
	SP	ESP

Регистры общего назначения

15 0	
CS	Код
SS	Стек
DS	Данные
ES	Данные
FS	Данные
GS	Данные

Сегментные регистры

31 16	15 0	
	IP	EIP
	FLAGS	EFLAGS

Регистры состояния и управления

Рисунок 1.1 – Программная модель процессора семейства x86

Регистры общего назначения без каких-либо ограничений могут использоваться для хранения операндов логических и арифметических операций, компонентов адреса, указателей на ячейки памяти:

EAX/AX/AH/AL (Accumulator register) – аккумулятор – применяется для хранения промежуточных данных;

EBX/BX/BH/BL (Base register) – базовый регистр – применяется для хранения базового адреса некоторого объекта в памяти;

ECX/CX/CH/CL (Count register) – регистр-счетчик – применяется в командах, производящих некоторые повторяющиеся действия;

EDX/DX/DH/DL (Data register) – регистр данных – так же, как и регистр EAX/AX/AH/AL, хранит промежуточные данные.

Следующие два регистра используются для поддержки цепочечных команд:

ESI/SI (Source Index register) – индекс источника – содержит текущий адрес элемента в цепочке-источнике;

EDI/DI (Destination Index register) – индекс приемника – содержит текущий адрес элемента в цепочке-приемнике.

Следующие регистры предназначены для работы со стеком:

ESP/SP (Stack Pointer register) – регистр указателя стека – содержит указатель вершины стека в сегменте стека. Этот регистр не следует использовать явно для хранения каких-либо операндов программы;

EBP/BP (Base Pointer register) – регистр указателя базы кадра стека – предназначен для организации произвольного доступа к данным внутри стека.

Сегментные регистры предназначены для аппаратной поддержки структурной организации программы в виде отдельных частей, называемых сегментами. Сегмент представляет собой независимый блок памяти фиксированного размера. Операционная система размещает сегменты программы в оперативной памяти, после чего помещает их адреса в соответствующие сегментные регистры.

МП поддерживает следующие типы сегментов:

сегмент кода – содержит машинные команды, для доступа к нему служит регистр CS (Code Segment register) – сегментный регистр кода;

сегмент стека – для доступа к нему служит регистр SS (Stack Segment register) – сегментный регистр стека;

сегмент данных – содержит обрабатываемые программой данные, для доступа к нему служит регистр DS (Data Segment register) – сегментный регистр данных;

дополнительные сегменты данных – их адреса должны содержаться в регистрах ES, GS, FS.

Регистры состояния и управления содержат информацию о состоянии МП и программы. К этим регистрам относятся: регистр флагов EFLAGS/FLAGS и регистр указатель команд EIP/IP. Используя эти регистры, можно получать информацию о результатах выполнения команд и влиять на состояние МП. На рисунке 1.2 показано содержимое регистра EFLAGS, а в таблице 1.1 – назначение отдельных флагов.

31	…	21	20	19	18	17	16	15	14	13	11	10	09	08	07	06	0	04	03	02	01	00
0	…	ID	VI	VI	A	V	R	0	N	IOP	O	D	IF	T	S	Z	0	A	0	P	1	C

Рисунок 1.2 – Содержимое регистра EFLAGS

Указатель команд EIP/IP содержит адрес следующей выполняемой команды относительно содержимого регистра CS. Регистр EIP/IP непосредственно недоступен программисту, но загрузка и изменение его значения производятся командами условных и безусловных переходов, вызова и возврата из процедур, а также вызова и возврата из прерываний.

С точки зрения размерности, МП аппаратно поддерживает следующие основные типы данных:

байт – восемь последовательно расположенных бит;

слово – два байта, имеющих последовательные адреса. Слово делится на младший байт и старший. Младший байт всегда хранится по меньшему адресу, который является адресом слова;

Таблица 1.1 – Назначение флагов регистра EFLAGS

Флаг	Название	Назначение
1	2	3
CF	Флаг переноса	= 1 – арифметическая операция произвела перенос из старшего бита результата = 0 – переноса не было
PF	Флаг паритета	= 1 – 8 младших разрядов результата содержат четное число единиц = 0 – нечетное
AF	Дополнительный флаг переноса	Для команд, работающих с BCD-числами: = 1 – перенос из разряда 3 в 4-й при сложении или заем в разряд 3 из 4-го при вычитании = 0 – не было переносов или заемов
ZF	Флаг нуля	= 1 – результат нулевой = 0 – результат ненулевой
SF	Флаг знака	= 1 – старший бит результата равен 1 = 0 – старший бит результата равен 0
TF	Флаг трассировки	При пошаговой работе МП: = 1 – МП генерирует прерывание с номером 1 после выполнения каждой машинной команды = 0 – обычная работа
IF	Флаг прерывания	= 1 – аппаратные прерывания разрешены = 0 – аппаратные прерывания запрещены
DF	Флаг управления	Определяет направление обработки цепочек: = 1 – от конца к началу = 0 – от начала к концу
OF	Флаг переполнения	= 1 – произошел перенос в старший или заем из старшего знакового бита результата = 0 – не было переноса или заема
IOPL	Уровень привилегий ввода-вывода	Для контроля доступа к командам ввода-вывода в защищенном режиме работы МП

Таблица 1.1 (продолжение)

1	2	3
VIP	Флаг отложенного виртуального прерывания	Появился в МП Pentium. = 1 – отложенное прерывание, используется при работе в V-режиме совместно с флагом VIF
ID	Флаг идентификации	Если программа может установить этот флаг, то МП поддерживает инструкцию CPUID
NT	Флаг вложенности задачи	Для фиксации в защищенном режиме работы МП того факта, что одна задача вложена в другую
RF	Флаг возобновления	Используется при обработке прерываний от регистров отладки
VM	Флаг виртуального 8086	= 1 – МП в режиме виртуального 8086 = 0 – МП в реальном или защищенном режиме
AC	Флаг контроля выравнивания	Разрешение контроля выравнивания при обращениях к памяти
VIF	Флаг виртуального прерывания	Появился в МП Pentium. В V-режиме является аналогом флага IF

двойное слово – четыре байта, расположенных по последовательным адресам. Двойное слово состоит из младшего слова и старшего. Младшее слово хранится по меньшему адресу, который является адресом двойного слова;

учетверенное слово – восемь байт, расположенных по последовательным адресам. Учетверенное слово делится на младшее двойное слово и старшее двойное слово. Младшее двойное слово хранится по меньшему адресу, который является адресом учетверенного слова;

128-битный упакованный тип данных – появился в МП Pentium III. Для работы с ним были введены специальные команды;

целый тип со знаком – двоичное значение со знаком размером 8/16/32 бита. Знак содержится соответственно в 7/15/31 бите (ноль – положительное

число, единица – отрицательное). Отрицательные числа представляются в дополнительном коде;

целый тип без знака – двоичное значение со знаком размером 8/16/32 бита;

ближний указатель – 32-разрядный логический адрес, представляющий собой относительное смещение в байтах от начала сегмента;

дальний указатель – 48-разрядный логический адрес, состоящий из двух частей: 16-разрядной сегментной части – селектора и 32-разрядного смещения;

цепочка – некоторый непрерывный набор байт, слов или двойных слов длиной до 4 Гбайт;

битовое поле – непрерывная последовательность бит. Каждый бит является независимым и может рассматриваться как отдельная переменная;

неупакованный двоично-десятичный тип – байтовое представление десятичной цифры от 0 до 9. Числа хранятся как байтовые значения без знака по одной цифре в каждом байте (в младшей тетраде);

упакованный двоично-десятичный тип – представление двух десятичных цифр от 0 до 9. Каждая цифра хранится в своей тетраде (старшая – в старшей, младшая – в младшей);

типы данных с плавающей точкой – специальные типы данных для обработки чисел с плавающей точкой в математическом сопроцессоре;

типы данных MMX-расширения (Pentium MMX/II) – представляют собой совокупность упакованных целочисленных элементов определенного размера;

типы данных XMM-расширения (Pentium III) – представляют собой совокупность упакованных элементов с плавающей точкой фиксированного размера.

Программа на языке ассемблера состоит из строк, содержащих следующие поля:

метка: команда/директива операнды ; комментарии

Все эти поля необязательны. Метка может быть любой комбинацией символов английского алфавита, цифр и символов «_», «$», «@», «?». Цифра не может быть первым символом метки. Большие и маленькие буквы по умолчанию не различаются, но различие можно включить, задав соответствующую опцию в командной строке ассемблера. В поле команды может располагаться команда процессора, которая будет транслирована в исполнимый код, или директива, которая не приводит к генерации кода, а управляет работой самого ассемблера. В поле операндов располагаются требуемые командой или директивой операнды. Текст от символа «;» не анализируется ассемблером и используется в качестве комментария.

Если метка располагается перед командой процессора, сразу после нее ставится символ «:», например:

```
some_loop:
    lodsw                    ;считать слово из строки в ax
    cmp ax, 7                ;если это 7 – выйти из цикла
loopne some_loop
```

Если метка стоит перед директивой – двоеточие не ставится. Рассмотрим директивы, работающие напрямую с метками и их значениями.

Определение переменных в общем виде записывается следующим образом:

name d value*

где d* – одна из псевдокоманд: db – определить байт, dw – определить слово (2 байта), dd – определить двойное слово (4 байта), df – определить

6 байт (дальний указатель), dq – определить учетверенное слово (8 байт), dt – определить 10 байт (80-битные типы данных, используемые FPU).

Поле значения может содержать одно или несколько чисел, символов строк (взятых в одиночные или двойные кавычки), операторов ? и операторов dup, разделенных запятыми, например:

```
text_string     db   'Hello world!', 0
number          dw   7
table           db   0,1,2,3,4,5,6,7,8,9
float_number    dd   3.5e7
```

Если вместо точного значения указан знак ?, переменная считается неинициализированной, например:

```
varl  dq    ?
```

Если требуется заполнить участок памяти повторяющимися данными, используется специальный оператор dup, имеющий формат

count   dup   (value)

Например, определение

```
table_512wdw          512          dup(?)
```

создает массив из 512 неинициализированных слов.

Структуры объявляются с помощью директивы struc, которая позволяет определить структуру данных аналогично структурам в языках высокого уровня:

struct_name   struc

fields

ends

где *fields* — любой набор псевдокоманд определения переменных или структур. В дальнейшем для создания объектов такой структуры в памяти используют имя структуры:

name struct name <values>

Для чтения и записи в элемент структуры используется оператор «.», например:

```
point   struc                        ;определение структуры
x       dw    0                      ;два слова со значениями
y       dw    0                      ;по умолчанию 0, 0
color   db    3   dup(?)             ;и три байта для цвета
ends
cur_point point <1,1,1,255,255,255>  ;инициализация
…
mov ax,   cur_point.x                ;обращение к слову x
```

Набор допустимых команд указывается в самом начале программы. По умолчанию ассемблер использует набор команд процессора 8086 и выдает сообщения об ошибках, если встречается команда, которую этот процессор не поддерживает. Для разрешения использования команд более новых процессоров предлагаются следующие директивы:

```
.186     ;команды 80186;
.286     ;непривилегированные команды 80286
.386     ;непривилегированные команды 80386
.486     ;непривилегированные команды 80486
.586     ;непривилегированные команды P5 (Pentium)
.686     ;непривилегированные команды P6 (Pentium)
.8087    ;команды FPU 8087
.287     ;команды FPU 80287
.387     ;команды FPU 80387
.487     ;команды FPU 80486
.587     ;команды FPU 80586
```

.MMX ;команды IA MMX

.K3D ;команды AMD 3D

Модель памяти в общем виде задается директивой

.model model_name, language, modifier,

где *model_name* – одно из значений, перечисленных в таблице 1.2; *language* – необязательный операнд, принимающий значения C, PASCAL, BASIC, FORTRAN, SYSCALL и STDCALL. Если он указан, ассемблер считает, что все процедуры рассчитаны на вызов из программ на соответствующем языке высокого уровня. *Modifier* – необязательный операнд, принимающий значения NEARSTACK (по умолчанию) или FARSTACK.

Директивы сегментации определяют сегменты программы. На практике чаще всего используют упрощенные директивы определения сегментов:

```
.stack size        ;сегмент стека в size байт
                   ;(по умолчанию 1024)
.data              ;обычный сегмент данных
.const             ;сегмент неизменяемых данных
.code              ;основной сегмент кода
```

Важной особенностью машинных команд является то, что они не могут манипулировать одновременно двумя операндами, находящимися в оперативной памяти. По этой причине возможны только следующие сочетания операндов в команде:

регистр – регистр;

регистр – память;

память – регистр;

регистр – непосредственный операнд;

память – непосредственный операнд.

Таблица 1.2 – Модели памяти

Модель	Описание
TINY	Код, данные и стек размещаются в одном и том же сегменте размером 64Кбайт
SMALL	Код – в одном сегменте, а данные и стек – в другом
COMPACT	Код – в одном сегменте, а данные могут располагаться в нескольких сегментах. Используются дальние указатели
MEDIUM	Код – в нескольких сегментах, а все данные – в одном. Для доступа к данным используется только смещение, а для вызова подпрограмм – команды дальнего вызова процедур
LARGE, HUGE	И код, и данные могут занимать несколько сегментов
FLAT	То же, что и TINY, но используются 32-разрядные адреса. Максимальный размер сегмента – 4 Мбайт

Из перечисленных сочетаний операндов наиболее часто употребляются регистр – память и память – регистр.

Рассмотрим систему команд микропроцессора x86.

1. Команда пересылки данных

MOV <адрес приемника> ,< адрес источника>

используется для пересылки данных длиной I или 2 байта из регистра в регистр, из регистра в основную память, из основной памяти в регистр, а также для записи в регистр или основную память данных, непосредственно записанных в команде. Все возможные пересылки представлены на рисунке 1.3.

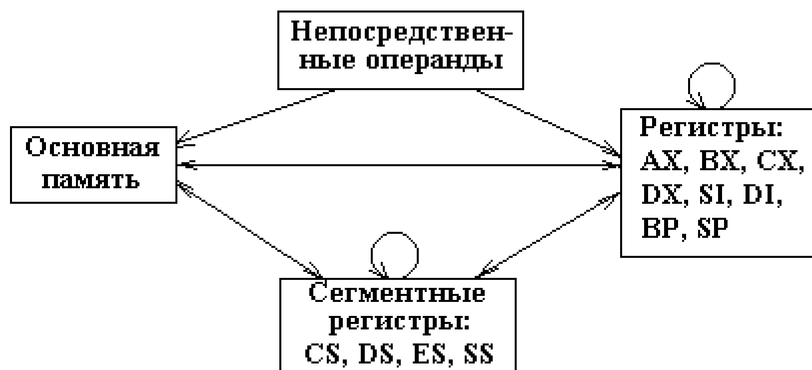


Рисунок 1.3 – Возможные пересылки

Примеры:

а) **mov ax, bx** – пересылка содержимого регистра **bx** в регистр **ax**;

б) **mov cx, exword** – пересылка 2 байт, расположенных в поле **exword**, из основной памяти в регистр **cx**;

в) **mov si, 1000** – запись числа 1000 в регистр **si**;

г) **mov word ptr [di+515], 4** – запись числа 4 длиной 2 байта в основную память по адресу **[di+515]**.

Для загрузки "прямого" адреса в сегментный регистр используются две команды пересылки:

```
mov   ax, code
mov   ds, ax
```

2. Команда обмена данных

XCHG <операнд 1> , <операнд 2>

организует обмен содержимого двух регистров (кроме сегментных) или регистра и поля основной памяти. Например:

xchg bx, cx – обмен содержимого регистров **bx** и **cx**.

3. Команда загрузки исполнительного адреса.

LEA < операнд 1 > , < операнд 2 >

вычисляет исполнительный адрес второго операнда и помещает его в поле, на которое указывает первый операнд. Приведем примеры:

а) **lea bx, exword** – в регистр **bx** загружается исполнительный адрес **exword**;

б) **lea bx, [di+10]** – в регистр **bx** загружается адрес 10-го байта относительно точки, на которую указывает адрес в регистре **di**.

4. Команды загрузки указателя

LDS < регистр > ,<операнд 2>

LES < регистр > ,< операнд 2>

Команда **LDS** загружает в регистры **DS** :< регистр> указатель (< адрес сегмента > : < исполнительный адрес >), расположенный по адресу, указанному во втором операнде.

Команда **LES** загружает указатель по адресу, расположенному во втором операнде, в регистры **ES**:< регистр> .

Например:

lds **si**, **exword**

т.е. слово (2 байта) по адресу **exword** загружается в **si**, а по адресу **exword**+ **2 –** в **ds**.

5. Команда записи в стек

PUSH < операнд>

организует запись в стек слова, адрес которого указан в операнде. Например;

push **dx** – запомнить содержимое регистра **dx** в стеке.

6. Команда восстановления из стека.

POP < операнд>

организует чтение из стека последнего слова и помещает его по адресу, указанному во втором операнде. Например:

pop **dx** – восстановить содержимое регистра **dx** из стека.

7. Команды сложения.

ADD <операнд 1> , <операнд 2>

ADC <операнд 1> , <операнд 2>

устанавливают флаги четности, знака результата, наличия переноса, наличия переполнения.

По команде **ADD** выполняется сложение двух операндов. Результат записывается по адресу первого операнда. По команде **ADC** также вы-

полнятся сложение двух операндов, но к ним добавляется еще значение, записанное в бите переноса, установленном предыдущей командой сложения.

Приведем пример сложения двух 32-разрядных чисел:

```
mov   ax, value1
add   value2, ax
mov   ax, value1+2
adc   value2+2, ax
```

Исходные числа находится в основной памяти по адресам **value1** и **value2**, а результат записывается по адресу **value1**.

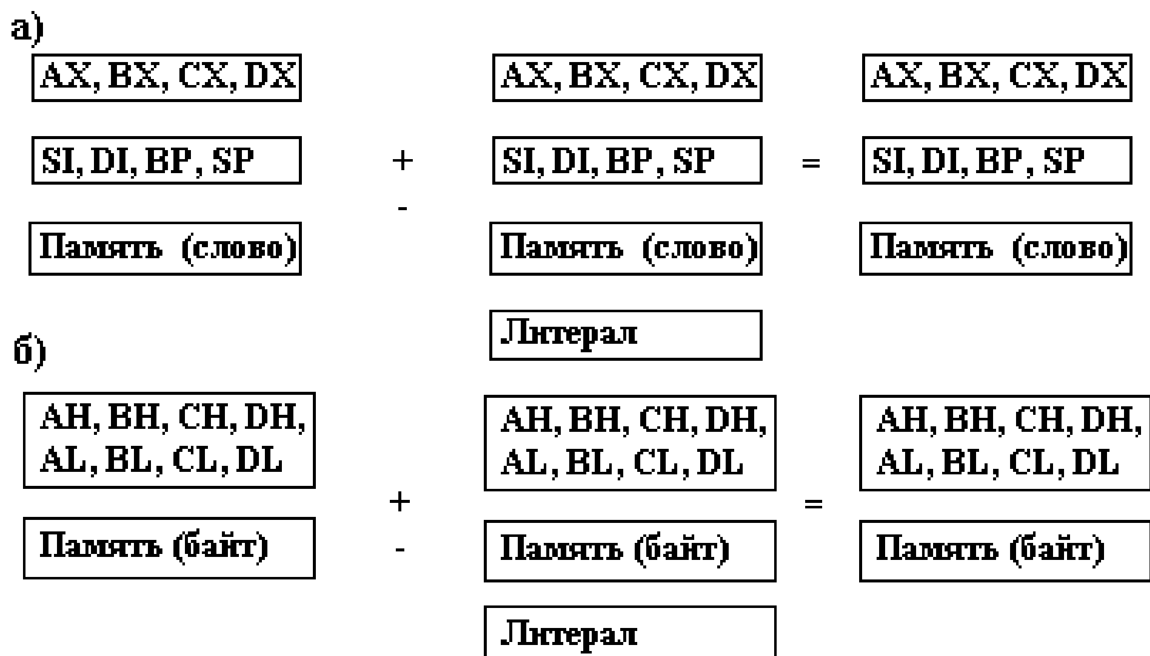


Рисунок 1.4 – Операция сложения:
а – операнды-слова,   б – операнды-байты

8. Команды вычитания

SUB <уменьшаемое-результат> , <вычитаемое>

SBB <уменьшаемое-результат>, <вычитаемое>

устанавливают флаги четности, знака результата, наличия заема, наличия переполнения.

При выполнении операции по команде **SUB** заем не учитывается, а по команде **SBB** - учитывается. Ограничения на местоположение операндов такие же, как и у команды сложения.

9. Команда изменения знака

NEG <операнд>

знак операнда изменяется на противоположный.

10. Команда добавления единицы

INC <операнд>

значение операнда увеличивается на единицу.

11. Команда вычитания единицы

DEC <операнд>

значение операнда уменьшается на единицу.

12. Команда сравнения

CMP <операнд 1> , < операнд 2>

выполняется операция вычитания без записи результата и устанавливаются признаки во флажковом регистре.

13. Команды умножения

MUL <операнд>

IMUL <операнд>

устанавливают флаги наличия переноса или переполнения.

По команде **MUL** числа перемножаются без учета, и по команде - **IMUL** – с учетом знака (в дополнительном коде).

На рисунке 1.5 (где а – операнды-слова, б - операнды-байты) приведены (один из сомножителей всегда расположен в регистре-аккумуляторе.

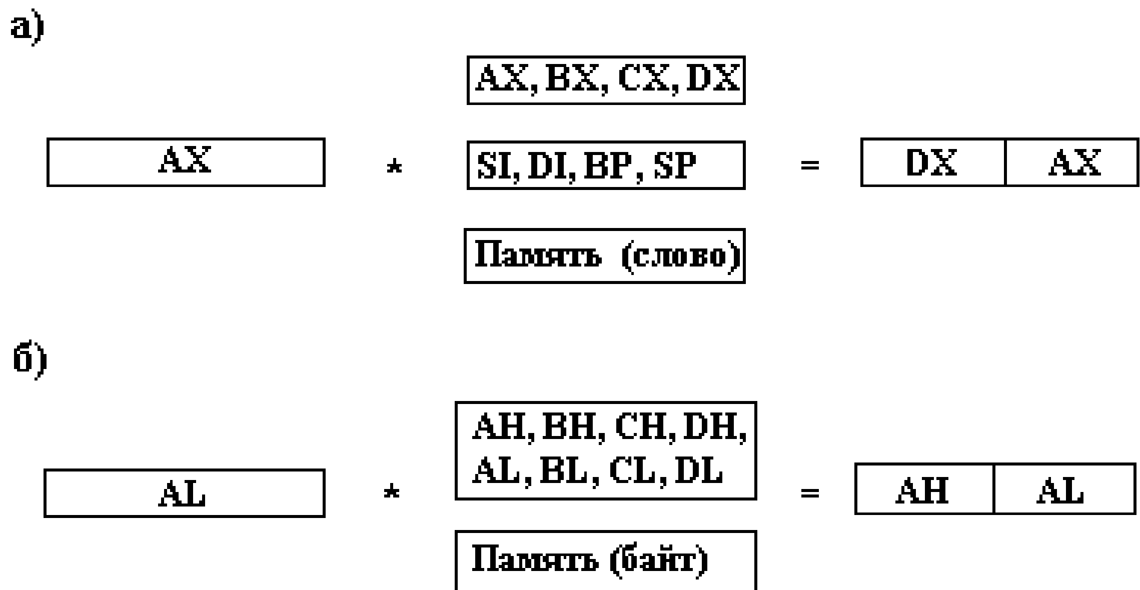


Рисунок 1.5 – Возможные способы размещения сомножителей и результата: а – операнды-слова, б – операнды-байты

Рассмотрим пример:

imul word ptr c

Здесь содержимое основной памяти по адресу "**c**" длиной слово умножается на содержимое регистра **ax**. Младшая часть результата операции записывается в регистр **ax**, а старшая часть – в регистр **dx**.

14. Команда деления

DIV <операнд-делитель>

IDIV <операнд-делитель>

По команде **DIV** операция деления выполняется без учета, а по команде **IDIV** – с учетом знака (в дополнительном коде).

На рис. 9 приведены способы размещения делимого, делителя и результа.

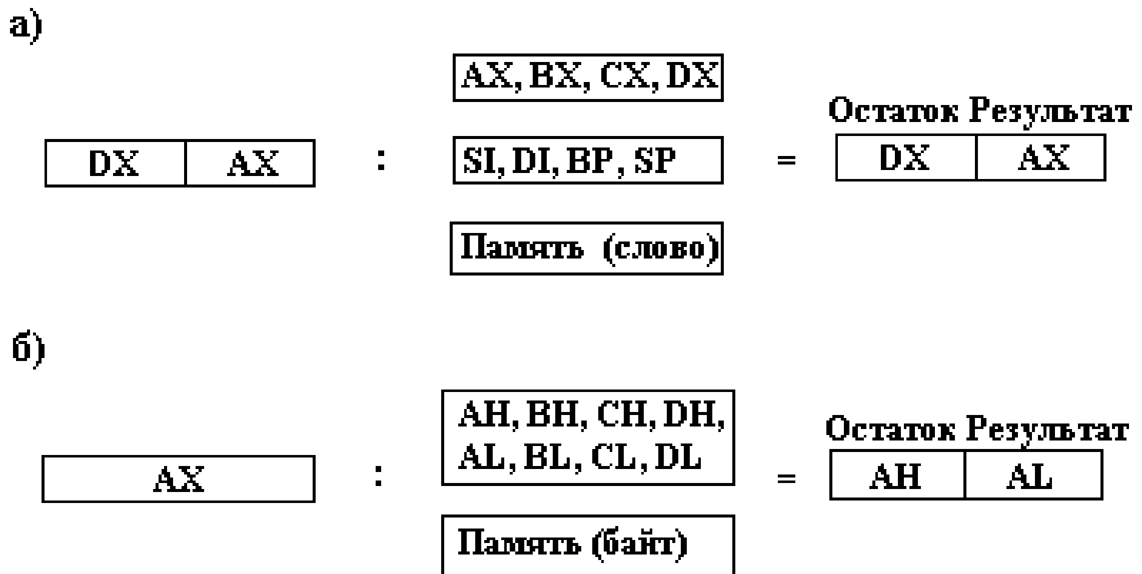


Рисунок 1.6 – Возможные способы размещения делимого, делителя и результата: а – операнды-слова, б – операнды-байты

15. Команда преобразования байта в слово, а слова – в двойное слово.

CBW

CWD

По команде **CBW** число из **al** переписывается в **ax** (дополнение выполняется знаковыми разрядами). Аналогично по команде **CWD** число из **ax** переписывается в два регистра **dx** и **ax**.

Традиционно первая программа для освоения нового языка программирования – программа, выводящая на экран текст «Hello world!». Наберите в любом текстовом редакторе следующий текст:

```
.model small
.stack
.data
Message db 'Hello world!', 13, 10, '$'
.code
begin:
```

```
mov  ax, @data             ;@data – адрес сегмента данных
mov  ds, ax                ;ds указывает на сегмент данных
mov  ah, 9                 ;функция  печати   строки   DOS
mov  dx, offset Message    ;dx указывает на Message
int  21h                   ;вывод на дисплей «Hello  world!»
mov  ah, 4ch               ;функция завершения программы в DOS
int  21h                   ;завершить программу
end  begin
```

Сохраните файл под именем hello.asm.

Прежде чем получить исполнимый файл, сначала нужно вызвать ассемблер, для того чтобы скомпилировать программу в объектный файл с именем hello.obj, набрав в командной строке следующую команду:

tasm hello

После успешного ассемблирования следующим шагом является компоновка. Для компоновки используется компоновщик tlink. В командной строке необходимо ввести

tlink hello

В результате при отсутствии ошибок будет получен исполнимый файл с именем hello.exe. Если его выполнить, на экране появится строка «Hello world!» и программа завершится.

Методика и порядок выполнения работы

1. Изучить программную модель микропроцессора x86, набор регистров, типы данных.

2. Изучить принципы построения программ на языке ассемблера, директивы, определение переменных, режимы адресации.

3. Разработать две программы под операционную систему DOS на языке ассемблера, соответствующие индивидуальному заданию из таблицы 1.3 (индивидуальное задание выбирается по номеру студента в журнале группы).

4. Защитить практическую работу. Защита заключается в построчном устном комментировании программы, запущенной под отладчиком TD.

Контрольные вопросы

1. Какое функциональное назначение у регистров микропроцессора x86?
2. Почему настройка сегментного регистра данных не производится непосредственно?
3. Какие способы адресации вы использовали в практической работе?
4. Каков размер сегментов Ваших программ?
5. Что происходит при компиляции?
6. Что содержит файл с расширением *.obj?
7. Какую модель памяти вы использовали и почему?

Таблица 1.3 – Задания на практическую работу № 1

№ варианта	Выражения, которые должны быть реализованы в программе (аргументы хранятся в памяти, результат сохраняется в сегмент данных)
1	y = (a – b)·c + 10; y = (4·a + b + c)/d
2	y = 4·a + b + 2·c; y = (4·a – b)/c
3	y = 2·b + b·a; y = ((a + 3·b)/c) + 4
4	y = (b + c)·a – 3·d; y = (a/c) + 4·b
5	y = b + mod(a,5); y = (3 + b)/a
6	y = 12·c – b – c; y = (b + a + c)/d
7	y = (15 – b)·c + a; y = ((13·a – b)/c) + a
8	y = (c + b + a)·3; y = (c + b - a)/a
9	y = (a + b)·5 + 9; y = (c + 7·b)/(a + d)
10	y = 3·a + b – 2·c; y = (4·a + mod(b,3))/c
11	y = 10·a + b – 3·c; y = (12·a – b)/c
12	y = (b – a)·c + d; y = ((14 + b)/c) – a
13	y = (a + c)·a + 14; y = ((17 + c)/a) + b·d
14	y = 3·c + 5·b – 4·a; y = (4·c + b·a)/d
15	y = 15·b + a – b; y = (b + a)/mod(c,7)
16	y = (a/b) + 3·c; y = (3·a – b) + c
17	y = b + a·c; y = (3·mod(a,2) + 2·b)/c
18	y = (a + b)/(c + 9); y = (a·5 – c)·b
19	y = 3·(b + c) – 2; y = 4·c – b + (a/3)
20	y = (7·a)/c + b; y = a·b – 3
21	y = 4·b + a·c; y = (4·c + b)/(a + d)
22	y = (b – 5·c)·a; y = (a·b – c)/d
23	y = c·a – b·d; y = (c·a + b·d)/11
24	y = c·b + a·5; y = ((c + a)/b +1) + 3·d
25	y = 7·a – 4·b + 9; y = (15 – a)/(3·b)
26	y = (3·b – 4)·a; y = (b/a) + (c/d)
27	y = 11·b + a – 10; y = (10 + b – c·a)/d

Практическая работа № 2

Команды передачи управления и работа со стеком

Цель работы: ознакомиться с принципами и командами передачи управления и работой со стеком. Научиться создавать программы с процедурами, передачей информации через стек и командами условного и безусловного перехода.

Краткие сведения из теории

1. Команда безусловного перехода.

JMP <адрес перехода>

имеет три модификации в зависимости от длины ее адресной части:

short – при переходе по адресу, который находится на расстоянии -128...127 байт относительно адреса данной команды (длина адресной части 1 байт);

near ptr – при переходе по адресу, который находится на расстоянии 32 Кбайта (-32768...32767 байт) относительно адреса данной команды (длина адресной части 2 байта);

far ptr – при переходе по адресу, который находится на расстоянии, превышающем 32 Кбайта (длина адресной части 4 байта).

При указании перехода к командам, предшествующим команде перехода, ассемблер сам определяет расстояние до метки перехода и строит адрес нужной длины. При указании перехода к последующим частям программы необходимо ставить указатели **short**, **near ptr** и **far ptr**.

В качестве адреса команды перехода используются метки трех видов:

а) < имя > : **nop** (**nop** - команда "нет операции");

б) < имя> **label near** (для внутрисегментных переходов);

в) <имя> **label far** (для внесегментных переходов).

Примеры:

а) **jmp short b** – переход по адресу **b**;

б) **jmp [bx]** – переход по адресу в регистре **bx** (адрес определяется косвенно);

в) **a : nop –** описание метки перехода "**a**";

г) **b label near** – описание метки перехода "**b**".

2. Команды условного перехода.

<мнемоническая команда> <адрес перехода>

Мнемоника команд условного перехода:

JZ – переход по "ноль";

JE – переход по "равно";

JNZ – переход по "не нуль";

JNE – переход по "не равно";

JL – переход по "меньше";

JNG, JLE – переход по "меньше или равно ";

JG – переход по "больше";

JNL, JGE – переход по "больше или равно ";

JA – переход по "выше" (беззнаковое больше);

JNA, JBE – переход по "не выше"(беззнаковое не больше);

JB – переход по "ниже" (беззнаковое меньше);

JNB, JAE – переход по"не ниже" (беззнаковое не меньше).

Все команды имеют однобайтовое поле адреса, следовательно, смещение не должно выходить за диапазон -128...127 байт. Если смещение выходит за указанные пределы, то используется специальный прием:

вместо программируется

```
jz   zero                 jnz  continue
                          jmp  zero
                continue: ...
```

3. Команды организации циклической обработки.

В качестве счетчика цикла во всех командах циклической обработки используется содержимое регистра **cx**.

1) Команда организации цикла.

LOOP < адрес перехода >

при каждом выполнении уменьшает содержимое регистра **cx** на единицу и передает управление по указанному адресу, если **cx** не равно 0:

```
        mov     cx,  loop_count          ;   загрузка счетчика
begin_loop:
        ;  ...   тело цикла ...
            loop   begin_loop
```

Примечание. Если перед началом цикла в регистр **cx** загружен 0, то цикл выполняется 35536 раз.

2) Команда перехода по обнуленному счетчику.

JCXZ <адрес перехода>

передает управление по указанному адресу, если содержимое регистра **cx** равно 0. Например:

```
        mov     cx,  loop_count          ;   загрузка счетчика
        jcxz    end_of_loop              ;   проверка счетчика
begin_loop:
    ;  ...   тело цикла ...
            loop    begin_loop
end_of_loop:    ...
```

3) *Команды организации цикла с условием.*

LOOPE <адрес перехода>

LOOPNE <адрес перехода>

уменьшают содержимое на единицу и передают управление по указанному адресу при условии, что содержимое **cx** отлично от нуля, но **LOOPE** дополнительно требует наличия признака "равно", а **LOOPNE** - "не равно", формируемых командами сравнения. Например:

```
        mov     cx, loop_count          ;   загрузка счетчика
        jcxz    end_of_loop             ;   проверка счетчика
begin_loop:
    ;  ...  тело цикла ...
        cmp     al, 100                 ; проверка содержимого al
        loopne  begin_loop              ; возврат в цикл, если cx≠0 и al≠100
end_of_loop:    ...
```

4. Команды вызова подпрограмм.

1) Команда вызова процедуры.

CALL <адрес процедуры>

осуществляет передачу управления по указанному адресу, предварительно записав в стек адрес возврата.

При указании адреса процедуры так же, как и при указании адреса перехода в командах безусловного перехода, возникает необходимость определить удаленности процедуры от места вызова:

а) если процедура удалена не более чем на -128...127 байт, то специальных указаний не требуется;

б) если процедура удалена в пределах 32 кбайт, то перед адресом процедуры необходимо указать **near ptr**,

в) если процедура подпрограмма удалена более, чем на 32 кбайта, то перед адресом процедуры необходимо записать **far ptr**.

Например:

call near ptr p – вызов подпрограммы "**p**".

Текст процедуры должен быть оформлен в виде:

< имя процедуры> **proc** < указатель удаленности>

... тело процедуры ...

<имя процедуры> **end**

Здесь указатель удаленности также служит дли определения длины адресов, используемых при обращении к процедуре: **near** – при использовании двухбайтовых адресов, **far** – при использовании четырехбайтовых адресов.

2) Команда возврата управления.

RET [<число>]

извлекает из стека адрес возврата и передает управление по указанному адресу.

Если в команде указано значение счетчика, то после восстановления адреса возврата указанное число добавляется к содержимому регистра-указателя стека. Последний вариант команды позволяет удалить из стека параметры, передаваемые в процедуру через стек.

Методика и порядок выполнения работы

В соответствии с вариантом напишите программу на языке ассемблера для вычисления значения y. Значения y1 и y2 необходимо вычислять в подпрограммах; параметры в подпрограммы передавать через стек; результат также возвращать через стек. Произведите трассировку программы для проверки правильности вычислений.

Исходные данные для вычисления выражений должны присутствовать в сегменте данных (a и x – числа со знаком длиной в байт). Результаты вычислений должны быть помещены в сегмент данных.

Таблица 2.1 – Задания на практическую работу № 2

№	Вычисления в основной программе	Первая процедура	Вторая процедура
1	**2**	**3**	**4**
1	$y = y1 + y2$	$y1 = \begin{cases} a + x, & if\ x > a \\ 2a - x, & if\ x \le a \end{cases}$	$y2 = \begin{cases} a + x, & if\ x > a \\ 2a - x, & if\ x \le a \end{cases}$
2	$y = y1 - y2$	$y1 = \begin{cases} x - 2, & if\ x \ge 2 \\ 8, & if\ x < 2 \end{cases}$	$y2 = \begin{cases} 4, & if\ x = 0 \\ a - x, & if\ x \ne 0 \end{cases}$
3	$y = y1 \times y2$	$y1 = \begin{cases} x - a, & if\ x > a \\ 5, & if\ x \le a \end{cases}$	$y2 = \begin{cases} a, & if\ a > x \\ a \times x, & if\ a \le x \end{cases}$
4	$y = y1 + y2$	$y1 = \begin{cases} 2 - x, & if\ x < 2 \\ a + 3, & if\ x \ge 2 \end{cases}$	$y2 = \begin{cases} a - 1, & if\ x < a \\ a \times x - 1, & if\ x \ge a \end{cases}$
5	$y = y1 - y2$	$y1 = \begin{cases} \lvert x \rvert, & if\ x < 0 \\ x - a, & if\ x \ge 0 \end{cases}$	$y2 = \begin{cases} a + x, & if\ x \bmod 3 = 1 \\ 7, & if\ x \bmod 3 \ne 1 \end{cases}$
6	$y = y1 + y2$	$y1 = \begin{cases} x \bmod 4, & if\ x > a \\ a, & if\ x \le a \end{cases}$	$y2 = \begin{cases} a \times x, & if\ x / a > 3 \\ x, & if\ x / a \le 3 \end{cases}$
7	$y = y1 + y2$	$y1 = \begin{cases} 4 - x, & if\ \lvert x \rvert < 3 \\ a + x, & if\ \lvert x \rvert \ge 3 \end{cases}$	$y2 = \begin{cases} 2, & if\ x - \text{÷åò íî å} \\ 2a - x, & if\ x - \text{í å÷åò íî å} \end{cases}$
8	$y = y1 + y2$	$y1 = \begin{cases} 4 \times x, & if\ x \le 4 \\ x - a, & if\ x > 4 \end{cases}$	$y2 = \begin{cases} 7, & if\ x - \text{í å÷åò íî å} \\ x / 2 + a, & if\ x - \text{÷åò íî å} \end{cases}$
9	$y = y1 \times y2$	$y1 = \begin{cases} a \times x, & if\ x \bmod 3 = 2 \\ 9, & if\ x \bmod 3 \ne 2 \end{cases}$	$y2 = \begin{cases} a - x, & if\ a > x \\ a + 2, & if\ a \le x \end{cases}$
10	$y = y1 - y2$	$y1 = \begin{cases} a + \lvert x \rvert, & if\ x > a \\ a - 7, & if\ x \le a \end{cases}$	$y2 = \begin{cases} a \times 3, & if\ a > 3 \\ 11, & if\ a \le 3 \end{cases}$
11	$y = y1 \bmod y2$	$y1 = \begin{cases} 10 + x, & if\ x > 1 \\ \lvert x \rvert + a, & if\ x \le 1 \end{cases}$	$y2 = \begin{cases} 2, & if\ x > 4 \\ x, & if\ x \le 4 \end{cases}$

Таблица 2.1 (продолжение)

1	2	3	4
12	$y = y1 / y2$	$y1 = \begin{cases} 15 + x, \ if \ x > 7 \\ \lvert a \rvert - 9, \ if \ x \le 7 \end{cases}$	$y2 = \begin{cases} 3, \ if \ x > 2 \\ \lvert x \rvert - 5, \ if \ x \le 2 \end{cases}$
13	$y = y1 \times y2$	$y1 = \begin{cases} 3 + x, \ if \ x = a \\ a - x, \ if \ x \ne a \end{cases}$	$y2 = \begin{cases} \lvert a \rvert, \ if \ x > 10 \\ \lvert a \rvert - x, \ if \ x \le 10 \end{cases}$
14	$y = y1 - y2$	$y1 = \begin{cases} 2x + a, \ if \ x > 2 \\ 2x + 1, \ if \ x \le 2 \end{cases}$	$y2 = \begin{cases} \lvert x \rvert + 1, \ if \ x > 0 \\ a - 1, \ if \ x \le 0 \end{cases}$
15	$y = y1 \bmod y2$	$y1 = \begin{cases} 8 + \lvert x \rvert, \ if \ x < 1 \\ \lvert a \rvert \times 2, \ if \ x \ge 1 \end{cases}$	$y2 = \begin{cases} 3, \ if \ x = a \\ a + 1, \ if \ x \ne a \end{cases}$
16	$y = y1 + y2$	$y1 = \begin{cases} 4 + x, \ if \ x \le 3 \\ a \times x, \ if \ x > 3 \end{cases}$	$y2 = \begin{cases} \lvert a \rvert - 2, \ if \ x > a \\ x, \ if \ x \le a \end{cases}$
17	$y = y1 - y2$	$y1 = \begin{cases} a + \lvert x \rvert, \ if \ x < 0 \\ x - a, \ if \ x \ge 0 \end{cases}$	$y2 = \begin{cases} 7, \ if \ x < 3 \\ a, \ if \ x \ge 3 \end{cases}$
18	$y = y1 \bmod y2$	$y1 = \begin{cases} 7 + x, \ if \ x < 3 \\ \lvert a \rvert + x, \ if \ x \ge 3 \end{cases}$	$y2 = \begin{cases} 1, \ if \ x > 5 \\ a + x, \ if \ x \le 5 \end{cases}$
19	$y = \lvert y1 \rvert + y2$	$y1 = \begin{cases} -5, \ if \ x > 4 \\ x - a, \ if \ x \le 4 \end{cases}$	$y2 = \begin{cases} \lvert a \rvert, \ if \ x > a \\ 9, \ if \ x \le a \end{cases}$
20	$y = y1 \times y2$	$y1 = \begin{cases} 2 \times x, \ if \ x < 5 \\ \lvert a \rvert + x, \ if \ x \ge 5 \end{cases}$	$y2 = \begin{cases} 3, \ if \ x < 0 \\ a + x, \ if \ x \ge 0 \end{cases}$
21	$y = y1 + \lvert y2 \rvert$	$y1 = \begin{cases} 3, \ if \ x \bmod 3 = 1 \\ x - a, \ if \ x \bmod 3 \ne 1 \end{cases}$	$y2 = \begin{cases} a / x, \ if \ x \ne 0 \\ 4, \ if \ x = 0 \end{cases}$
22	$y = y1 - y2$	$y1 = \begin{cases} \lvert a \rvert + \lvert x \rvert, \ if \ x < 0 \\ a \times x, \ if \ x \ge 0 \end{cases}$	$y2 = \begin{cases} 3, \ if \ x = a \\ a - x, \ if \ x \ne a \end{cases}$
23	$y = y1 + y2$	$y1 = \begin{cases} 2 \times x, \ if \ \lvert x \rvert > 4 \\ 4 + a, \ if \ \lvert x \rvert \le 4 \end{cases}$	$y2 = \begin{cases} 9, \ if \ x = 0 \\ a / x, \ if \ x \ne a \end{cases}$
24	$y = y1 \times y2$	$y1 = \begin{cases} x, \ if \ x \bmod 4 \ne 2 \\ a + x, \ if \ x \bmod 4 = 2 \end{cases}$	$y2 = \begin{cases} a - x, \ if \ x < a \\ a \times x, \ if \ x \ge a \end{cases}$
25	$y = y1 / y2$	$y1 = \begin{cases} 12, \ if \ x < 12 \\ x + 1, \ if \ x \ge 12 \end{cases}$	$y2 = \begin{cases} 2, \ if \ x > 2 \\ a + x, \ if \ x \le 2 \end{cases}$

Контрольные вопросы

1. Содержимое каких регистров и сегментов изменяют команды call, ret, jmp, cmp?

2. Как производится условный переход?

3. Как производится вызов процедуры?

4. Что изменяет содержимое флажкового регистра?

5. Как можно использовать флажковый регистр?

6. Какие команды изменяют содержимое регистра ip?

7. Почему обмен через стек является более предпочтительным?

8. Какие способы адресации вы использовали в практической работе?

Практическая работа № 3

Обработка массивов

Цель работы: научиться создавать и обрабатывать одномерные и двумерные массивы на языке ассемблера процессора x86.

Краткие сведения из теории

Массив во внутреннем представлении – это последовательность элементов в памяти, например:

A dw 10,13,28,67,0,-1 ; массив из 6 чисел длиной слово.

Программирование обработки выполняется с использованием адресного регистра, в котором хранится либо адрес текущего элемента, либо его смещение относительно начала массива. При переходе к следующему элементу адрес (или смещение) увеличивается на длину элемента.

Пример. Написать процедуру, выполняющую суммирование массива из 10 чисел размером слово.

Вариант 1 (используется адрес):

```
summas   proc
         mov   ax, 0
lea      bx, MAS
mov      cx, 10
CYCL:    add   ax, [bx]
add      bx, 2
loop     CYCL
ret
summas   endp
```

Вариант 2 (используется смещение):

```
summas   proc
mov      ax, 0
mov      bx, 0
mov      cx, 10
CYCL:    add   ax, MAS [bx]
add      bx, 2
loop     CYCL
ret
summas   endp
```

Второй вариант позволяет получать более наглядный код и потому является предпочтительным.

В том случае, если элементы просматриваются непоследовательно, адрес элемента может рассчитываться по его номеру: $A_{исп}$ = $A_{нача-}$

ла+(<номер>-1)*<длина элемента>. Полученный по формуле адрес записывается в один из адресных регистров (**BX, BP, DI, SI**) и используется для доступа к элементу.

Пример. Написать процедуру, которая извлекает из массива, включающего 10 чисел размером слово, число с номем n (n≤10).

```
n_mas       proc
mov   bx, N     ;  номер числа
dec   bx        ;  вычитаем 1
sal   bx, 1     ;  умножили на длину (сдвинули влево на 1)
mov   ax,  MAS [bx]  ; результат в ax
ret
      n_mas         endp
```

Рассмотрим моделирование матриц.

Значения матрицы могут располагаться в памяти по строкам и по столбцам. Для определенности будем считать, что матрица расположена в памяти построчно.

При моделировании обработки матрицы следует различать просмотр по строкам, просмотр по столбцам, просмотр по диагоналям и произвольный доступ.

Просмотр по строкам иногда может выполняться так, как в одномерном массиве (без учета перехода от одной строки к другой).

Пример. Написать процедуру определения максимального элемента матрицы A(3,5).

```
maxmatr     proc
mov     bx,  0     ; смещение 0
mov     cx,  14    ;  счетчик цикла
```

```
mov      ax,  A      ;  заносим первое число
CYCL:         cmp      ax,  A[bx+2]  ;  сравниваем числа
jge       NEXT       ;  если больше, то перейти к следующему
mov      ax,  A[bx+2]    ; если меньше, то запомнить
NEXT:          add       bx,  2      ;  переходим к следующему числу
loop     CYCL
ret       ;  результат в ax
maxmatr        endp
```

Просмотр по строкам при необходимости фиксировать завершение строки и просмотр по столбцам выполняются в двойном цикле: по строкам – во внешнем цикле, по столбцам – во внутреннем или наоборот. В этом случае обычно отдельно формируются смещения строки и столбца.

Пример. Определить сумму максимальных элементов столбцов матрицы A(3,5).

```
maxmatr        proc
mov      ax, 0       ; обнуляем сумму
mov      bx,  0      ; смещение элемента столбца в строке
mov      cx,  5     ;  количество столбцов
CYCL1:        push     cx         ;  сохраняем счетчик
               mov      cx, 2      ;  счетчик элементов в столбце
               mov      dx, A[bx]  ; заносим первый элемент столбца
 mov      si, 10      ; смещение второго элемента столбца
CYCL2:        cmp      dx,  A[bx]+ [si]  ; сравниваем
 jge       NEXT   ; если больше или равно – к следующему
 mov      dx,  A[bx]+[si]  ;  если меньше, то сохранили
NEXT:          add       si,  10    ;  переходим к следующему элементу
loop      CYCL2   ; цикл по элементам столбца
```

```
add      ax,  dx  ;  просуммировали макс. элемент
pop      cx         ;  восстановили счетчик
add      bx,  2   ;  перешли к следующему столбцу
loop     CYCL1    ;  цикл по столбцам
ret               ;   результат в ax
maxmatr      endp
```

Методика и порядок выполнения работы

Разработать программу для обработки одномерных массивов (минимум шесть элементов) и программу для обработки двумерных массивов (4 на 4) на языке ассемблера процессора x86.

Моделирование одномерных массивов:

1. Вычислить сумму элементов массива
2. Вычислить среднее арифметическое значение элементов массива
3. Определить максимальный элемент массива и его порядковый номер
4. Вычислить минимальный элемент массива и его номер
5. Найти максимальный и минимальный элементы массива и поменять их местами
6. Расположить в массиве сначала положительные, а затем отрицательные элементы массива
7. Определить сумму элементов массива, кратных трем
8. Переписать в массив Y подряд положительные элементы массива X
9. Переписать подряд в массив Y положительные и в массив Z отрицательные элементы массива X
10. Определить сумму элементов массива, кратных пяти
11. Определить сумму положительных элементов массива
12. Переписать в массив Y элементы массива X, кратные трем

13. Записать на место положительных элементов нули

14. Подсчитать число положительных и отрицательных элементов в массиве

15. Определить номера элементов кратных 3

16. Создать массив из остатков от целочисленного деления числа 100 на простые числа из диапазона от 2 включительно до 31

17. Произвести поэлементное сложение двух массивов

18. Создать массив из остатков от целочисленного деления массива (14 15 12 8 21 16) на 11

19. Вычислить поразрядную конъюнкцию элементов массива

20. Перемножить элементы массива на константу

21. Найти наибольший по абсолютному значению элемент массива

22. Найти наименьший по абсолютному значению элемент массива и запомнить его позицию

23. Записать в массив X компоненты массива Y, которые больше 3, но меньше 20

24. Запомнить порядковые номера всех четных элементов пассива

25. Получить дополнения элементов заданного массива

Моделирование двумерных массивов:

1. Вычислить и запомнить сумму и число положительных элементов каждого столбца матрицы.

2. Вычислить и запомнить суммы и числа элементов каждой строки матрицы.

3. Вычислить суммы элементов матрицы, находящихся под главной диагональю и на ней

4. Вычислить сумму элементов матрицы, находящихся над главной диагональю

5. Записать на место отрицательных элементов нули
6. Найти определитель матрицы
7. Транспонировать матрицу
8. Найти в каждой строке матрицы максимальный и минимальный элементы и поместить их на место первого и последнего элемента строки соответственно
9. Записать на место отрицательных элементов матрицы нули, а на место положительных – единицы
10. Для целочисленной матрицы найти для каждой строки число элементов кратных пяти и наибольший из полученных результатов
11. Найти в каждой строке наибольший элемент и поменять его местами с элементом главной диагонали
12. Найти наибольший и наименьший элементы матрицы и поменять их местами
13. Найти строку с наибольшей и наименьшей суммой элементов
14. Упорядочить по возрастанию элементы каждой строки матрицы
15. Записать на место чисел под главной диагональю нули
16. Записать на место чисел над главной диагональю единицы
17. Инвертировать элементы над и под главной диагональю
18. Найти первые три минора матрицы
19. Сложить две матрицы
20. Перемножить две матрицы
21. Записать на место элементов под и над главной диагональю результат поразрядной конъюнкции с элементом главной диагонали
22. Найти обратную матрицу
23. Поменять местами первую и последнюю строчку матрицы
24. Поменять местами второй и предпоследний столбец матрицы

25. Заменить элементы главной диагонали на единицы, а остальные элементы – на нули

Контрольные вопросы

1. Какие регистры изменяет команды loop?
2. Как организовать цикл с заданным числом итераций?
3. Как организовать циклы с неопределенным числом итераций?
4. Как задаются массивы на языке ассемблера?
5. Какие способы адресации вы использовали в практической работе?
6. Как задать и изменять в программе индексы массива?
7. Сколько байт в памяти занимает команда push ax?

Практическая работа № 4

Система прерываний IBM совместимых ЭВМ. Ввод-вывод информации.

Цель работы: Получить навыки составления программ на языке ассемблера с вводом данных с клавиатуры и выводом результатов вычислений на экран.

Краткие сведения из теории

Прерыванием называется временное переключение процессора на выполнение некоторой заранее определенной последовательности команд, после завершения которой процесс выполнения программы возобновляется. Прерывания в IBM совместимых ЭВМ (см. рисунок 4.1) могут инициироваться как специальными сигналами микропроцессора (внутренние), так и внешними сигналами, например, от внешних устройств (внешние).

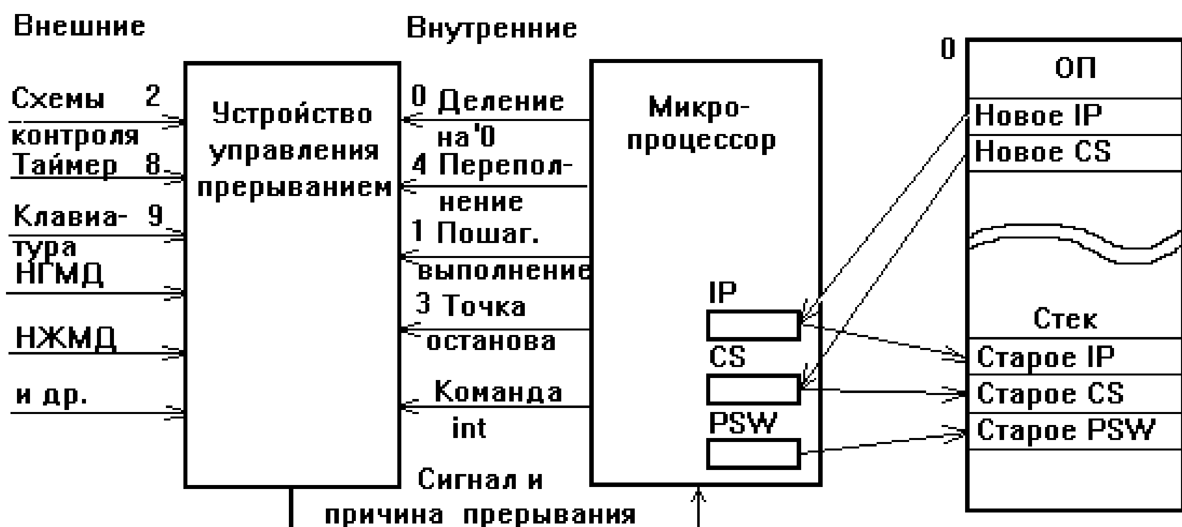


Рисунок 4.1 – Система обработки прерываний.

Внешние сигналы на прерывание поступают на специальные микросхемы – контроллеры прерываний 8259A, имеющие 8 уровней приоритета. Начиная с PC AT микроЭВМ включает два контроллера, что позво-

ляет увеличить количество уровней приоритета до 16. Уровни приоритета определяются аппаратно в зависимости от места подключения внешнего устройства.

Как правило, самый высокий приоритет имеет системный таймер, затем следует клавиатура, что обеспечивает оперативное управление микроЭВМ, далее идут прочие внешние устройства. Завершают список обычно накопители на гибких магнитных дисках и устройство печати.

Прерывания, поступающие через контроллеры прерываний, также называют *аппаратными* в отличие от остальных, носящих название *программных*.

Выполнение всех прерываний, кроме прерывания 2, можно запретить, установив в 0 флаг IF флажкового регистра. Прерывание 2 в связи с этим носит название *немаскируемого* и используется для того, чтобы выполнять обработку сигналов, наличие которых нельзя игнорировать, например, сигнала о недопустимом изменении напряжения в сети питания.

Адреса программ-обработчиков прерываний хранятся в специальной области основной памяти, которая обычно располагается с 0-го адреса и занимает 1кБ, то есть может содержать адреса 256 программ (каждый адрес занимает 4 байта). Местоположение адреса нужного обработчика в области векторов прерываний определяется по типу (номеру) прерывания:

$$A=4*N,$$ где N - номер прерывания.

При наличии сигнала прерывания выполняется следующая последовательность действий:

1) проверяется установка флажка IF (для немаскируемых прерываний этот пункт игнорируется): 1 – прерывания разрешены, 0 – прерывания запрещены;

2) если прерывание разрешено, то после завершения выполнения текущей команды слово состояния программы (PSW), хранящееся во флажковом регистре микропроцессора, значение сегментного регистра кодов (CS) и значение счетчика команд (IP) заносятся в стек;

3) из области векторов прерываний в регистры CS и IP помещается адрес программы обработки прерываний.

После чего начинается выполнение программы обработки прерывания.

По завершении этой программы значения PSW, CS и IP восстанавливаются из стека, и выполнение прерванной программы возобновляется.

Использование большинства прерываний определяется конкретным программным обеспечением и соответственно для каждой ПЭВМ будет своим. Однако существуют номера прерываний, закрепленные за соответствующими функциями.

1. *Прерывания микропроцессора (0H-6H)*:

0 – прерывание от схем контроля микропроцессора – «Деление на 0»;

1 – прерывание специального режима работы микропроцессора, устанавливаемого, если флажок TF=1 – «Пошаговое выполнение»;

2 – немаскируемое прерывание;

3 – прерывание микропроцессора, осуществляемого при обнаружении адреса останова – «Точка останова»;

4 – инициируется по команде INTO, используемой после выполнения арифметической операции – «Переполнение»;

5 – печать содержимого экрана – инициируется нажатием клавиши Print Screen.

2. *Прерывания микроконтроллера прерываний (7H-0FH, 70H-77H)*:

8 – прерывание от таймера;

9 – прерывание от клавиатуры;

0BH – COM2;

0CH – COM1;

0EH – прерывание от НГМД (дискеты);

0FH – прерывание от печатающего устройства;

70H – прерывание от часов реального времени;

76H – прерывание от НЖМД (жесткий диск);

3. *Процедуры BIOS (10H-1AH, 33H, 41H):*

10H – управление дисплеем;

11H – определение конфигурации ПЭВМ;

12H – определение объема памяти ПЭВМ;

13H – управление дисковой памятью;

14H – управление асинхронной связью;

16H – управление клавиатурой;

17H – управление печатающим устройством;

1AH – управление часами реального времени.

4. *Процедуры пользователя (1BH и 1CH):*

1BH – возможность подключения при обнаружении Ctrl-Break;

1CH – возможность подключения к обработке кванта таймера.

5. *Указатели системных таблиц (1DH-1FH, 41H):*

1DH – таблица параметров видео;

1EH – таблица параметров дискеты;

1FH – таблица символов для графического режима;

41H – таблица параметров жесткого диска.

6. ***Прерывания DOS (20H- 3FH):***

20H – нормальное завершение программы и возврат управления DOS;

21H – вызов диспетчера функций DOS;

22H – адрес пользовательской программы обработки нормального завершения программы;

23H – адрес пользовательской программы обработки завершения по Ctrl-Break;

24H – адрес пользовательской программы обработки завершения по ошибке;

25H – абсолютное чтение секторов с диска;

26H – абсолютная запись секторов на диск;

27H – завершение программы с сохранением в памяти.

7. Прерывания, зарезервированные для пользователей (60H-66H, 0F0H-0FEH).

При организации ввода-вывода помимо самой операции необходимо осуществить ряд дополнительных действий, например, проверить готовность устройства. В связи с этим для типовых устройств разработаны стандартные программы организации ввода-вывода, которые вызываются по команде прерывания **int 21h**.

В таблице 4.1 приведен перечень основных функций, реализуемых подпрограммами ввода-вывода, и их коды. Код функции должен передаваться в подпрограмму в регистре **AH**.

Примеры:

а)
```
        mov    ah, 1    ; номер функции
        int    21h      ; ввод символа: символ в AL
```

```
б)   mov   ah, 2   ; номер функции
     mov   dl, 'A'
     int   21h     ; вывод символа из DL
```

Таблица 4.1 – Основные функции ввода-вывода

Код функции	Функция
01	Ввод с клавиатуры одного символа в регистр AL (с проверкой на Ctrl-Break, с ожиданием, с эхо)
02	Вывод одного символа на экран дисплея из регистра DL (с проверкой на Ctrl-Break)
06	Непосредственный ввод-вывод: ввод в регистр AL (без ожидания, без эхо, без проверки на Ctrl-Break, регистр DL должен содержать 0FFH), вывод из регистра DL (без проверки на Ctrl-Break)
07	Ввод в регистр AL (без проверки на Ctrl-Break, с ожиданием, без эхо)
08	Ввод в регистр AL (с проверкой на Ctrl-Break, с ожиданием, без эхо)
09	Вывод строки на экран (DS:DX – адрес строки, которая должна завершаться символом "$")
10(0Ah)	Ввод строки в буфер (DS:DX – адрес буфера, первый байт которого должен содержать размер буфера, после ввода - второй байт содержит количество введенных символов)
11(0Bh)	Чтение состояния клавиатуры (если буфер пуст, то AL=0, иначе AL=0FFh)

```
в)   lea   dx, STRING  ; адрес буфера ввода
     mov   ah, 0Ah     ; номер функции
     int   21h     ; ввод строки:  во втором байте буфера - количество
     ...           ; введенных символов, далее в буфере символы
  STRING  db  50, 51 dup (?)
г)   lea   dx, MSG  ; адрес выводимой строки
     mov   ah, 9    ; номер функции
```

```
        int     21h     ; вывод строки
        ...
MSG     db      'Пример вывода', 13, 10, '$'
```

Ввод с клавиатуры

Средства DOS

DOS предоставляет набор функций для чтения данных с клавиатуры, которые используют стандартное устройство ввода STDIN, так что можно использовать в качестве источника данных файл или стандартный вывод другой программы.

Функция DOS 0Ah – Считать строку символов из STDIN в буфер

Ввод: AH = 0Ah

DS:DX = адрес буфера

Вывод: Буфер содержит введенную строку

Для вызова этой функции надо подготовить буфер, первый байт которого содержит максимальное число символов для ввода (1-255) с учетом символа 0dh, прекращающего процесс ввода, а содержимое, если оно задано, может использоваться как подсказка для ввода. Если первый байт равен нулю, то вызов функции игнорируется и программа продолжает выполнение без ожидания ввода строки. При наборе строки обрабатываются клавиши Esc, F3, F5, BS, Ctrl-C/Ctrl-Break и т.д., как при наборе команд DOS (то есть Esc начинает ввод сначала, F3 восстанавливает подсказку для ввода, F5 запоминает текущую строку как подсказку, Backspace стирает предыдущий символ). После нажатия клавиши Enter строка (включая последний символ CR (0Dh) записывается в буфер, начиная с третьего байта. Во второй байт записывается длина реально

введенной строки без учета последнего CR (максимальная длина строки – 254 символа).

Функция 0Ah предоставляет удобный, но ограниченный способ ввода данных. Чаще всего используют функции посимвольного ввода, позволяющие контролировать отображение символов на экране, реакцию программы на функциональные и управляющие клавиши и т.д.

Функция DOS 01h – Считать символ из STDIN с эхом, ожиданием и проверкой на Ctrl-Break

Ввод: AH = 01h

Вывод: AL = ASCII-код символа или 0. Если AL = 0, второй вызов этой функции возвратит в AL расширенный ASCII-код символа

При чтении с помощью этой функции введенный символ автоматически немедленно отображается на экране (посылается в устройство STDOUT – так что его можно перенаправить в файл). При нажатии Ctrl-C или Ctrl-Break выполняется команда INT 23h. Если нажата клавиша, не соответствующая какому-нибудь символу (стрелки, функциональные клавиши Ins, Del и т.д.), то в AL возвращается 0, и функцию надо вызвать еще один раз, чтобы получить расширенный ASCII-код.

Функция DOS 08h – Считать символ из STDIN без эха, с ожиданием и проверкой на Ctrl-Break

Ввод: AH = 08h

Вывод: AL = код символа

Функция DOS 07h – Считать символ из STDIN без эха, с ожиданием и без проверки на Ctrl-Break

Ввод: AH = 07h

Вывод: AL = код символа

Функция DOS 06h – Считать символ из STDIN без эха, без ожидания и без проверки на Ctrl-Break

Ввод: AH = 07h

DL = 0FFh

Вывод: ZF = 1, если не была нажата клавиша, и AL = 00

ZF = 0, если клавиша была нажата. В этом случае AL = код символа

Кроме перечисленных функций могут потребоваться и некоторые служебные функции DOS для работы с клавиатурой.

Функция DOS 0Bh – Проверить состояние клавиатуры

Ввод: AH = 0Bh

Вывод: AL = 0, если не была нажата клавиша

AL = 0FFh, если была нажата клавиша

Эту функцию удобно использовать перед функциями 01, 07 и 08, чтобы не ждать нажатия клавиши. Кроме того, вызов этой функции позволяет проверить, не считывая символ с клавиатуры, была ли нажата комбинация клавиш Ctrl-Break; если это произошло, выполнится прерывание 23h.

Функция DOS 0Ch – Очистить буфер и считать символ

Ввод: AH = 0Ch

AL = Номер функции DOS (01, 06, 07, 08, 0Ah)

Вывод: Зависит от вызванной функции

Функция 0Ch очищает буфер клавиатуры, так что следующая функция чтения символа будет ждать ввода с клавиатуры, а не использовать нажатый ранее и еще не обработанный символ. Например, именно эта функция

используется для считывания ответа на вопрос «Уверен ли пользователь в том, что он хочет отформатировать диск?».

Средства BIOS

BIOS предоставляет больше возможностей по сравнению с DOS для считывания данных и управления клавиатурой. Например, функциями DOS нельзя определить нажатие комбинаций клавиш типа Ctrl-Alt-Enter или нажатие двух клавиш Shift одновременно, DOS не может определить момент отпускания нажатой клавиши, и наконец, в DOS нет аналога функции C ungetch(), помещающей символ в буфер клавиатуры, как если бы его ввел пользователь. Все это можно осуществить, используя различные функции прерывания 16h и операции с байтами состояния клавиатуры.

INT 16h, AH = 0, 10h, 20h – Чтение символа с ожиданием

Ввод: AH = 00h (83/84-key), 10h (101/102-key), 20h (122-key)

Вывод: AL = ASCII-код символа, 0 или префикс скан-кода

AH = скан-код нажатой клавиши или расширенный ASCII-код

Каждой клавише на клавиатуре соответствует так называемый скан-код, соответствующий только этой клавише. Этот код посылается клавиатурой при каждом нажатии и отпускании клавиши и обрабатывается BIOS (обработчиком прерывания INT 9). Прерывание 16h дает возможность получить код нажатия, не перехватывая этот обработчик. Если нажатой клавише соответствует ASCII-символ, то в AH возвращается код этого символа, а в AL – скан-код клавиши. Если нажатой клавише соответствует расширенный ASCII-код, в AL возвращается префикс скан-кода (например, E0 для серых клавиш) или 0, если префикса нет, а в AH – расширенный ASCII-код. Функция 00H обрабатывает только комбинации, использующие клавиши 84-клавишной клавиатуры, 10h обрабатывает все

101-105-клавишные комбинации, 20h – 122-клавишные. Тип клавиатуры можно определить с помощью функции 09h прерывания 16h, если она поддерживается BIOS (поддерживается ли эта функция, можно узнать с помощью функции C0h прерывания 15h).

INT 16h, AH = 1, 11h, 21h – Проверка символа

Ввод: AH = 01h (83/84-key), 11h (101/102-key), 21h (122-key)

Вывод: ZF = 1, если буфер пуст

ZF = 0, если в буфере присутствует символ, в этом случае

AL = ASCII-код символа, 0 или префикс скан-кода

AH = скан-код нажатой клавиши или расширенный ASCII-код

Символ остается в буфере клавиатуры, хотя некоторые BIOS удаляют символ из буфера при обработке функции 01h, если он соответствует расширенному ASCII-коду, отсутствующему на 84-клавишных клавиатурах.

INT 16h, AH = 05h – Поместить символ в буфер клавиатуры

Ввод: AH = 05h

CH = скан-код

CL = ASCII-код

Вывод: AL = 00, если операция выполнена успешно

AL = 01h, если буфер клавиатуры переполнен

AH модифицируется многими BIOS

Обычно можно поместить 0 вместо скан-кода в CH, если функция, которая будет выполнять чтение из буфера, будет использовать именно ASCII-код.

INT 16h, AH = 02h, 12h, 22h – Считать состояние клавиатуры

Ввод: AH = 02h (83/84-key), 12h (101/102-key), 22h (122-key)

Вывод: AL = байт состояния клавиатуры 1

AH = байт состояния клавиатуры 2 (только для функций 12h и 22h)

Байт состояния клавиатуры 1 (этот байт всегда расположен в памяти по адресу 0000h:0417h или 0040h:0017h):

Бит 7: Ins включена

Бит 6: CapsLock включена

Бит 5: NumLock включена

Бит 4: ScrollLock включена

Бит 3: Alt нажата (любая Alt для функции 02h, часто только левая Alt для 12h/22h)

Бит 2: Ctrl нажата (любая Ctrl)

Бит 1: Левая Shift нажата

Бит 0: Правая Shift нажата

Байт состояния клавиатуры 2 (этот байт всегда расположен в памяти по адресу 0000h:0418h или 0040h:0018h):

Бит 7: SysRq нажата

Бит 6: CapsLock нажата

Бит 5: NumLock нажата

Бит 4: ScrollLock нажата

Бит 3: Правая Alt нажата

Бит 2: Правая Ctrl нажата

Бит 1: Левая Alt нажата

Бит 0: Левая Ctrl нажата

Оба этих байта постоянно располагаются в памяти, так что вместо вызова прерывания часто удобнее просто считывать значения напрямую.

Помимо этих двух байт BIOS хранит в своей области данных и весь клавиатурный буфер, к которому также можно обращаться напрямую. Буфер занимает 16 слов с 0h:041Eh по 0h:043Dh включительно, причем по адресу 0h:041Ah лежит адрес (ближний) начала буфера, то есть адрес, по которому располагается следующий введенный символ, а по адресу 0h:041Ch лежит адрес конца буфера, так что если эти два адреса равны, буфер пуст. Буфер действует как кольцо: если начало буфера – 043Ch, а конец – 0420h, то в буфере находятся три символа по адресам 043Ch, 041Eh и 0420h. Каждый символ хранится в виде слова – того же самого, которое возвращает функция 10h прерывания INT 16h. В некоторых случаях (если) буфер размещается по другим адресам, тогда адрес его начала хранится в области данных BIOS по адресу 0480h, а конца – по адресу 0482h. Прямой доступ к буферу клавиатуры лишь немногим быстрее, чем вызов соответствующих функций BIOS, и для приложений, требующих максимальной скорости, таких как игры или демо-программы, используют управление клавиатурой на уровне портов ввода-вывода.

Вывод на экран в текстовом режиме

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).

Эта функция при выводе на экран обрабатывает некоторые управляющие символы – вывод символа BEL (07h) приводит к звуковому сигналу, символ BS (08h) приводит к движению курсора влево на одну позицию, символ HT (09h) заменяется на несколько пробелов, символ LF (0Ah) опускает курсор на одну позицию вниз, и CR (0Dh) приводит к переходу на начало текущей строки.

Если в ходе работы этой функции была нажата комбинация клавиш Ctrl-Break, вызывается прерывание 23h, которое по умолчанию осуществляет выход из программы.

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.

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)

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.

INT 29h – Быстрый вывод символа на экран

Ввод: AL = ASCII-код символа

В большинстве случаев INT 29h просто немедленно вызывает функцию BIOS «вывод символа на экран в режиме телетайпа», так что никаких преимуществ, кроме экономии байт при написании как можно более коротких программ, она не имеет.

Средства BIOS

Функции DOS вывода на экран позволяют перенаправлять вывод в файл, но не позволяют вывести текст в любую позицию экрана и не позволяют изменить цвет текста. DOS предполагает, что для более тонкой работы с экраном программы должны использоваться видеофункции BIOS. BIOS (базовая система ввода-вывода) – это набор программ, расположенных в постоянной памяти компьютера, которые выполняют его загрузку сразу после включения и обеспечивают доступ к некоторым устройствам, в частности к видеоадаптеру. Все функции видеосервиса BIOS вызываются через прерывание 10h. Рассмотрим функции, которые могут быть полезны для вывода текстов на экран (полностью видеофункции BIOS описаны в приложении 2).

Выбор видеорежима

BIOS предоставляет возможность переключения экрана в различные текстовые и графические режимы. Режимы отличаются друг от друга разрешением (для графических) и количеством строк и столбцов (для текстовых), а также количеством возможных цветов.

INT 10h, AH = 00 – Установить видеорежим

Ввод: AL = номер режима в младших 7 битах

Вывод: Обычно никакого, но некоторые BIOS (Phoenix и AMI) помещают в AL 30H для текстовых режимов и 20h для графических

Вызов этой функции приводит к тому, что экран переводится в выбранный режим. Если старший бит AL не установлен в 1, экран очищается. Номера текстовых режимов — 0, 1, 2, 3 и 7. 0 и 1 — 16-цветные режимы 40x25 (с 25 строками по 40 символов в строке), 2 и 3 — 16-цветные режимы 80x25, 7 — монохромный режим 80x25. Мы не будем пока рассматривать графические режимы, хотя функции вывода текста на экран DOS и BIOS могут работать и в них.

INT 10h, AH = 4Fh, AL = 02 – Установить SuperVGA-видеорежим

Ввод: BX = номер режима в младших 13 битах

Вывод: AL = 4Fh, если эта функция поддерживается

AH = 0, если переключение произошло успешно

AH = 1, если произошла ошибка

Если бит 15 регистра BX установлен в 1, видеопамять не очищается. Текстовые режимы, которые можно вызвать с использованием этой функции: 80x60 (режим 108h), 132x25 (109h), 132x43 (10Ah), 132x50 (10Bh), 132x60 (10Ch).

Видеорежим, используемый в DOS по умолчанию, — текстовый режим 3.

Управление положением курсора

INT 10h, AH = 02 – Установить положение курсора

Ввод: AH = 02

BH = номер страницы

DH = строка

DL = столбец

С помощью этой функции можно установить курсор в любую позицию экрана, и дальнейший вывод текста будет происходить из этой позиции. Отсчет номера строки и столбца ведется от верхнего левого угла экрана (символ в левой верхней позиции имеет координаты 0, 0). Номера страниц 0 – 3 (для режимов 2 и 3)и 0 – 7 (для режимов 1 и 2) соответствуют области памяти, содержимое которой в данный момент отображается на экране. Можно вывести текст в неактивную в настоящий момент страницу, а затем переключиться на нее, чтобы изображение изменилось мгновенно.

INT 10h, AH = 03 – Считать положение и размер курсора

Ввод: AH = 03

BH = номер страницы

Вывод: DH, DL = строка и столбец текущей позиции курсора

CH, CL = первая и последняя строки курсора

Возвращает текущее состояние курсора на выбранной странице (каждая страница использует собственный независимый курсор).

Вывод символов на экран

Каждый символ на экране описывается двумя байтами — ASCII-кодом символа и байтом атрибута, указывающим цвет символа и фона, а также является ли символ мигающим.

Атрибут символа:

Бит 7: символ мигает (по умолчанию) или фон яркого цвета (если его действие было переопределено видеофункцией 10h).

Биты 6 – 4: цвет фона.

Бит 3: символ яркого цвета (по умолчанию) или фон мигает (если его действие было переопределено видеофункцией 11h).

Биты 2 – 0: цвет символа.

Цвета кодируются в битах, как показано в таблице 4.2.

Таблица 4.2 – Атрибуты символов

	Обычный цвет	**Яркий цвет**
000b	черный	темно-серый
001b	синий	светло-синий
010b	зеленый	светло-зеленый
011b	голубой	светло-голубой
100b	красный	светло-красный
101b	пурпурный	светло-пурпурный
110b	коричневый	желтый
111b	светло-серый	белый

INT 10h, AH = 08 – Считать символ и атрибут символа в текущей позиции курсора

Ввод: AH = 08

BH = номер страницы

Вывод: AH = атрибут символа

AL = ASCII-код символа

INT 10h, AH = 09 – Вывести символ с заданным атрибутом на экран

Ввод: AH = 09

BH = номер страницы

AL = ASCII-код символа

BL = атрибут символа

CX = число повторений символа

С помощью этой функции можно вывести на экран любой символ, включая даже символы CR и LF, которые обычно интерпретируются как конец строки. В графических режимах CX не должен превышать число позиций, оставшееся до правого края экрана.

INT 10h, AH = 0Ah – Вывести символ с текущим атрибутом на экран

Ввод: AH = 0Ah

BH = номер страницы

AL = ASCII-код символа

CX = число повторений символа

Эта функция также выводит любой символ на экран, но в качестве атрибута символа используется атрибут, который имел символ, находившийся ранее в этой позиции.

INT 10h, AH = 0Eh – Вывести символ в режиме телетайпа

Ввод: AH = 0Eh

BH = номер страницы

AL = ASCII-код символа

Символы CR (0Dh), LF (0Ah), BEL (7) интерпретируются как управляющие символы. Если текст при записи выходит за пределы нижней строки, экран прокручивается вверх. В качестве атрибута используется атрибут символа, находившегося в этой позиции.

INT 10h, AH = 13h – Вывести строку символов с заданными атрибутами

Ввод: AH = 13h

AL = режим вывода:

бит 0 – переместить курсор в конец строки после вывода

бит 1 – строка содержит не только символы, но также и атрибуты, так что каждый символ описывается двумя байтами: ASCII-код и атрибут

биты 2 – 7 зарезервированы

CX = длина строки (только число символов)

BL = атрибут, если строка содержит только символы

DH,DL = строка и столбец, начиная с которых будет выводиться строки ES:BP = адрес начала строки в памяти

Функция 13h выводит на экран строку символов, интерпретируя управляющие символы CR (0Dh), LF (0Ah), BS (08) и BEL (07).

Методика и порядок выполнения работы

практическая работа включает два задания. Задание № 1 одинаковое для всех студентов группы и выполняется самостоятельно без защиты для получения знания и навыка ввода-вывода информации на языке ассемблера процессора x86. Задание № 2 включает индивидуальное задание, которое студент выполняет самостоятельно и защищает преподавателю.

Задание № 1

Набрать в текстовом редакторе программу и сохранить в файле с расширением .asm:

```
.model small
.stack 100h
.data
```

```
max db 255
len db ?
stroka db 255 dup(?)
  db ?
ent db 13, 10, '$'
.code
start:      mov ax, @data
      mov ds, ax
      xor ax, ax
      mov ah, 0ah
      lea dx, max
      int 21h
      mov bl, len
      mov bh, 0
      mov [bx + stroka +1], '$'
      lea dx, ent
      mov ah, 09h
      int 21h
      lea dx, stroka
      int 21h
      mov ah, 4ch
      int 21h
end start
```

Полученный файл откомпилировать с помощью tasm.exe, а с помощью компоновщика tlink.exe получить исполняемый файл. В отладчике выполнить следующие пункты:

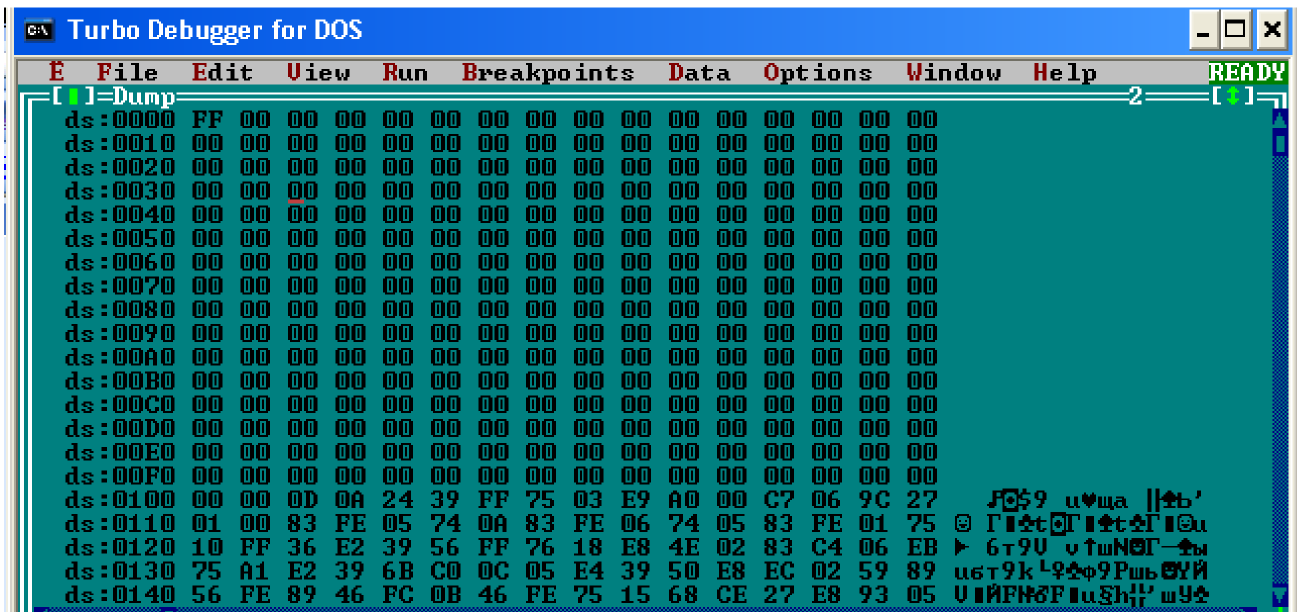
1. Провести анализ сегмента данных (Dump)

```
.data
max db 255                    ; максимальная длина строки
len db ?                  ; длина после ввода строки
stroka db 255 dup(?)          ; буфер для строки
   db ?                       ; резервный байт
ent db 13, 10, '$'            ; перевод строки
```

Поскольку в данной программе для ввода символов используется функция DOS 0Ah, то необходимо подготовить буфер, первый байт которого (max) содержит максимальное число символов для ввода (255). Второй неинициализированный байт (len) зарезервирован для записи длины введенной строки без учета символа CR (0Dh). Буфер (stroka) начинается с третьего байта (адрес DS:0002). Далее следует резервный байт, который будет использован для записи признака конца строки ('$'), если буфер будет полностью заполнен.

Сегмент данных, начиная со смещения [0102], включает байты 13 (соответствует символу CR (0Dh) – переход на начало текущей строки), 10 (символ LF (0Ah) – опускает курсор на одну позицию вниз), 24 (соответствует символу конца строки, '$').

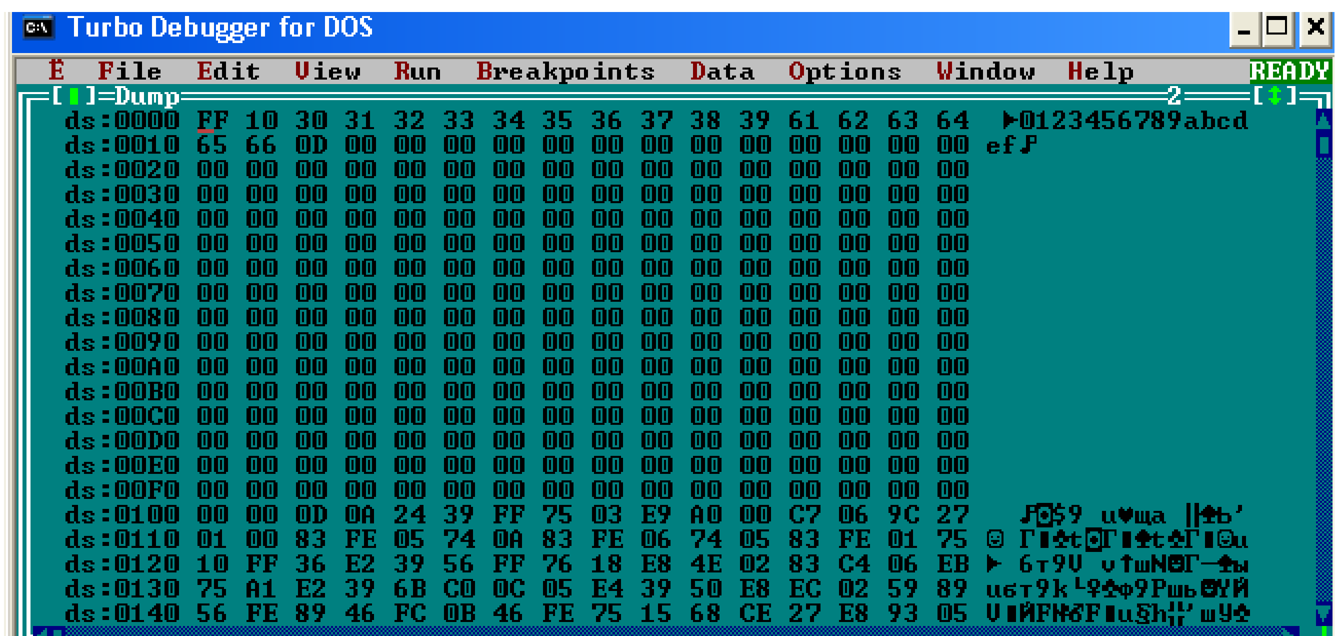
Начальное содержимое сегмента данных:



2. Проследить процесс ввода строки с клавиатуры (изменение содержимого буфера для строки и регистров)

```
mov ah, 0ah        ; записываем код функции DOS
lea dx, max        ; указываем на начало буфера
int 21h            ; осуществляется прерывание
```

Содержимое сегмента данных после ввода строки «0123456789abcdef»:



Байт DS:0000 – также содержит длину строки

Байт DS:0001 – содержит число введенных символов $((16)_{10} = (10)_{16})$

Байты с DS:0002 по DS:0011 – содержат ASCII-коды введенных символов

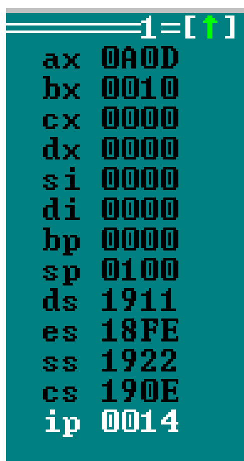
Байт DS:0012 – содержит код символа CR

3. Проследить запись признака конца строки ('$') в конец введенной строки символов:

Две команды для записи в регистр bx длины введенной строки:

mov bl, len ; bl = 10h

mov bh, 0 ; bx = 0010h

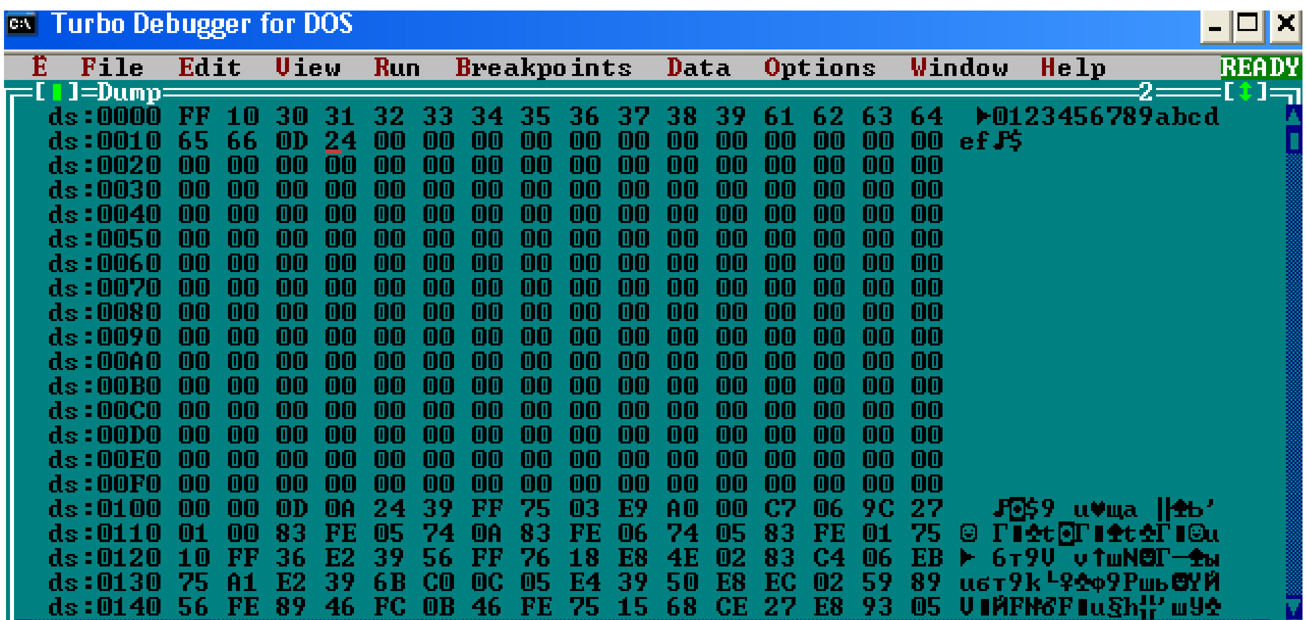


Следующей командой записываем по адресу DS:0013 символ '$', что необходимо для определения конца строки при ее выводе на экран:

mov [bx + stroka +1], '$' ; адрес определяется суммой [bx + 0003],

; т.к. адрес [stroka] = 0002

Сегмент данных:



4. Проследить перевод строки, заданный кодом (Alt+F5):

```
lea dx, ent
mov  ah, 09h
int 21h
```

5. Проследить вывод строки введенных символов (Alt+F5):

```
lea dx, stroka
int 21h
```

Задание № 2

Ввод данных с клавиатуры и вывод результатов вычислений на экран осуществляется средствами и функциями в соответствии с вариантом задания (см. таблица 4.3).

Числа, вводимые с консоли в символьном виде, должны быть преобразованы в соответствующее им внутреннее двоичное представление для дальнейших вычислений. Числа, которые должны быть выведены на

консоль, необходимо преобразовать из внутреннего двоичного представления в число в символьном виде, формат записи которого соответствует правилам требуемой системы счисления: десятичной или шестнадцатеричной.

Задание 2.1. Выполнить задание практической работы № 2, но с вводом данных A и X с клавиатуры и выводом результата Y экран. При вводе A на экране должно появится сообщение «Введите значение A:», при вводе X – сообщение «Введите значение X:». Разрядность чисел A и X принять 8 бит, а результата Y – 2 байта. При выводе на экран результата вычисления необходимо предварительно вывести сообщение «Результат вычисления Y=» (само значение можно вывести на следующей строке).

Таблица 4.3 – Индивидуальные задания на практическую работу № 4

Вариант	Основания системы счисления для ввода информации	Основания системы счисления для вывода информации	Функция ввода	Функция вывода
1	16 п	10	DOS	DOS
2	16 ст	10	DOS	BIOS
3	16 ст+п	10	BIOS	DOS
4	16 ст	10	BIOS	BIOS
5	16 ст+п	16 п	DOS	DOS
6	16 ст	16 ст	DOS	BIOS
7	16 ст+п	16 п	BIOS	DOS
8	16 п	16 ст	BIOS	BIOS
9	10	10	DOS	DOS
10	10	10	DOS	BIOS
11	10	10	BIOS	DOS
12	10	10	BIOS	BIOS
13	10	16 ст	DOS	DOS
14	10	16 п	DOS	BIOS

15	10	16 ст	BIOS	DOS
16	10	16 п	BIOS	BIOS
17	10	10	DOS	DOS
18	16 п	10	DOS	BIOS
19	16 ст	10	BIOS	DOS
20	16 ст+п	10	BIOS	BIOS
21	16 ст+п	16 ст	DOS	DOS
22	16 ст+п	16 п	DOS	BIOS
23	16 п	16 п	BIOS	DOS
24	16 ст	16 ст	BIOS	BIOS
25	16 ст+п	10	DOS	DOS
26	10	10	DOS	BIOS
27	10	10	BIOS	DOS
28	10	10	BIOS	BIOS

Задание 2.2. Выполнить задание 1 практической работы № 3, но с вводом значений одномерного массива и выводом результата вычислений на экран. Сообщения о вводе элементов массива и выводе результатов обязательны и должны начинаться с новой строки.

Рекомендации: Преобразование шестнадцатеричной ***цифры*** в ASCII-код соответствующего символа можно осуществить с помощью следующего кода (необходимо при выводе результата вычислений в шестнадцатеричной системе счисления на консоль):

```
cmp al, 10
sbb al, 69h
das
```

Команда сравнения (**cmp al, 10**) содержимого регистра al с 10, результатом которой является изменение флагов. В данном случае нам

представляет интерес содержимое флага CF, который равен 1, если al < 10, и равен 0, если al > 10, потому что следующей команда осуществляет вычитание с учетом переноса.

Результатом вычитания (**sbb al, 69h**) являются числа 96h ÷ 9Fh, если в регистре al хранились числа из диапазона 0 ÷ 9, и числа 0A1h ÷ 0A6h, если в регистре хранились числа 0Ah ÷ 0Fh.

Следующей командой (**das**) осуществляется BCD-коррекция после вычитания, посредством вычитания 66h из первой группы чисел, переводя их в 30h ÷ 39h, и 60h из второй группы чисел, переводя их в 41h ÷ 46h (см. таблицу ASCII-кодов).

Символы ASCII. Номера строк соответствуют первой цифре в шестнадцатеричном коде символа, номера столбцов – второй, так что, например, код большой латинской буквы A – 41h.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		☺	☻	♥	♦	♣	♠	•	◘	○	◙	♂	♀	♪	♫	☼
1	►	◄	↕	‼	¶	§	▬	↨	↑	↓	→	←	∟	↔	▲	▼
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	⌂

Пример 1: Написать программу для вычисления y = a·x с выводом результата на экран. Вводить числа в десятичной системе счисления (диапазон 0 – 99).

```
.model small
.486
```

```
.stack 100h
.data
c db 0Ah
mess1 db 'Введите значение A:', 13, 10, '$'
mess2 db 13, 10, 'Введите значение X:', 13, 10, '$'
mess3 db 13, 10, 'Результат вычисления Y=', 13, 10, '$'
.data?
a dw ?
x dw ?
y dw ?
.code
ent1 proc                    ; процедура ввода символов
xor ax, ax
xor dx, dx
mov ah, 1h
int 21h
mov dl, al
sub dl, 30h
int 21h
xchg dl, al
sub dl, 30h
mul c
add al, dl
ret
ent1 endp
print proc                   ; процедура вывода символа на экран
cmp al, 10
```

```
sbb al, 69h
das
mov dl, al
mov ah, 2
int 21h
ret
print endp
start:      mov ax, @data
    mov ds, ax
    xor ax, ax
    lea dx, mess1
    mov ah, 9
    int 21h             ; вывод сообщения «Введите значение А:»
    call ent1
    mov a, ax
    lea dx, mess2
    mov ah, 9
    int 21h             ; вывод сообщения «Введите значение Х:»
    call ent1
    mov x, ax
    mul a
    mov y, ax
    lea dx, mess3       ; вывод сообщения «Результат вычисления Y=»
    mov ah, 9
    int 21h
    mov ax, y
    xchg ah, al
```

```
        mov dh, al
        and dh, 0fh              ; младший символ старшего байта в dh
        shr al, 4                ; старший символ старшего байта в al
        call print
        mov al, dh
        call print
        mov ax, y
        mov dh, al
        and dh, 0fh              ; младший символ младшего байта в dh
        shr al, 4                ; старший символ младшего байта в al
        call print
        mov al, dh
        call print
        mov ax, 4c00h
        int 21h
end start
```

Пример 2: Получить дополнительные коды элементов массива. Вводить числа в шестнадцатеричной системе счисления с учетом регистра. Вывести в шестнадцатеричной системе счисления.

```
.model small
.386
.stack 100h
.data
max db 255
len db ?
```

```
stroka db 255 dup(?)
  db ?
mas db 127 dup (?)
ent db 13, 10, '$'
mess1 db 'Введите массив',13, 10, '$'
mess2 db 'Результат преобразования:',13, 10, '$'
.code
start:      mov ax, @data
        mov ds, ax
;Ввод строки – значения элементов массива по две цифры (0 ÷ F) на число
        lea dx, mess1
        mov ah, 9
        int 21h
        xor ax, ax
        mov ah, 0ah
        lea dx, max
        int 21h
;Инициализация регистров перед преобразованием
;«ASCII-коды → численное значение »
        xor ax, ax
        xor dx, dx
        xor cx, cx
        mov cl, len
        shr cx, 1
        lea bx, stroka
        xor si, si
loop1:      mov dl, [bx]      ;преобразование из ASCII-кодов в 16-ричный
```

```
                    ; код и запись в mas
        sub dl, 30h
        cmp dl, 9h
        jle M1
        sub dl, 7h
        cmp dl, 0fh
        jle M1
        sub dl, 20h
M1:     shl dl, 4h
        mov al, [bx + 1]
        sub al, 30h
        cmp al, 9h
        jle M2
        sub al, 7h
        cmp al, 0fh
        jle M2
        sub al, 20h
M2:     add dl, al
        mov mas [si], dl
        inc si
        add bx, 2
        loop loop1
        ; Преобразование в дополнительный код
        mov cl, len
        shr cx, 1
        xor si, si
loop2:      neg mas[si] ;
```

```
        inc si
        loop loop2
        ; Преобразование из 16-ричного представления в ASCII-код и запись
        ;в stroka
        lea bx, stroka
        mov cl, len
        shr cx, 1
        xor si, si
loop3:  mov al, mas [si]
        mov dh, al
        and dh, 0fh
        shr al, 4
        cmp al, 10
        sbb al, 69h
        das
        mov [bx], al
        mov al, dh
        cmp al, 10
        sbb al, 69h
        das
        mov [bx + 1], al
        add bx, 2
        inc si
        loop loop3

        mov bl, len
        mov bh, 0
```

```
        mov [bx + stroka +1], '$'
        lea dx, ent
        mov ah, 09h
        int 21h
        lea dx, mess2
        mov ah, 9
        int 21h
        ; Вывод результата на экран. Код 20h соответствует пробелу.
        lea bx, stroka
        mov cl, len
        shr cx, 1
loop4:  mov al, [bx]
        int 29h
        mov al, [bx + 1]
        int 29h
        mov al, 20h
        int 29h
        add bx, 2
        loop loop4
        mov ah, 4ch
        int 21h
end start
```

Контрольные вопросы

1. Что такое прерывание?
2. Какие виды прерываний Вам известны?
3. В чем отличие между вводом-выводом средствами DOS и BIOS?

4. Как произвести ввод с клавиатуры?
5. Что характерно для трассировочного прерывания?
6. Что такое слово состояния программы и где оно используется?
7. Какими средствами осуществляется посимвольный ввод информации?
8. Как произвести перевод символьного представления цифр в двоичное и обратно?

Литература

Основная литература

1. Новиков Ю.В. Основы микропроцессорной техники. — Электрон. текст. дан.— М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — Режим доступа : http ://www.iprbookshop.ru/52207. — ЭБС «IPRbooks», по паролю.

2. Водовозов А.М. Микроконтроллеры для систем автоматики : Учеб. пособие. — Электрон. текст. дан. — М. : Инфра-Инженерия, 2016. — Режим доступа : http ://www.iprbookshop.ru/51727.— ЭБС «IPRbooks», по паролю

Дополнительная литература

1. Гуров В.В. Архитектура микропроцессоров [Электронный ресурс]/ Гуров В.В.— Электрон. текстовые данные.— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 115 с.— Режим доступа: http://www.iprbookshop.ru/56313.html.— ЭБС «IPRbooks»

2. Аблязов Р.З. Программирование на ассемблере на платформе x86-64 [Электронный ресурс]/ Аблязов Р.З.— Электрон. текстовые данные.— Саратов: Профобразование, 2017.— 304 с.— Режим доступа: http://www.iprbookshop.ru/63951.html.— ЭБС «IPRbooks»

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

НЕВИННОМЫССКИЙ ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)

Методические рекомендации к организации самостоятельной работы по дисциплине

«Управляющие микропроцессорные комплексы»

Направление подготовки 15.04.04

«Автоматизация технологических процессов и производств» Направленность (профиль)

«Информационно-управляющие системы» Форма обучения - очно-заочная

Год начала обучения 2022

Реализуется в 2 семестре

Невинномысск 2022

Методические указания предназначены для студентов направления 15.04.04 Автоматизация технологических процессов и производств и других технических специальностей. Они содержат рекомендации по организации самостоятельных работ студента на правления 15.04.04 для дисциплины «Управляющие микропроцессорные комплексы».

Методические указания разработаны в соответствии с требованиями ФГОС ВО в части содержания и уровня подготовки выпускников направления 15.04.04 Автоматизация технологических процессов и производств.

Код	Формулировка
ПК-3	Способность: составлять описание принципов действия и конструкции устройств, проектируемых технических средств и систем автоматизации, управления, контроля, диагностики и испытаний технологических процессов и производств общепромышленного и специального назначения для различных отраслей национального хозяйства; проектировать их архитектурно-программные комплексы

Составитель канд. техн. наук Кочеров Ю. Н.

Ответственный редактор канд. техн. наук Д.В. Болдырев

Содержание

1 Подготовка к лабораторным занятиям.. 4

2 Подготовка к практическим занятиям.. 4

3 Самостоятельное изучение темы. Конспект 6

4 Комплект заданий для выполнения контрольной работы.......................... 9

4.1 Сокеты и протоколы 9

4.2 Процесс установления соединения 14

4.3 Порядок выполнения контрольной работы...19

Контрольные вопросы...20

1 Подготовка к лабораторным занятиям

Для того чтобы лабораторные занятия приносили максимальную пользу, необходимо помнить, что упражнение и решение задач проводятся по рассмотренному на лекциях материалу и связаны, как правило, с детальным разбором отдельных вопросов лекционного курса. Следует подчеркнуть, что только после усвоения лекционного материала с определенной точки зрения (а именно с той, с которой он излагается на лекциях) он будет закрепляться студентом на лабораторных занятиях как в результате обсуждения и анализа лекционного материала, так и с помощью решения проблемных ситуаций, задач. При этих условиях студент не только хорошо усвоит материал, но и научится применять его на практике, а также получит дополнительный стимул (и это очень важно) для активной проработки лекции.

При самостоятельном решении задач нужно обосновывать каждый этап решения, исходя из теоретических положений курса. Если студент видит несколько путей решения проблемы (задачи), то нужно сравнить их и выбрать самый рациональный. Полезно до начала вычислений составить краткий план решения проблемы (задачи). Решение проблемных задач или примеров следует излагать подробно, вычисления располагать в строгом порядке, отделяя вспомогательные вычисления от основных. Решения при необходимости нужно сопровождать комментариями, схемами, чертежами и рисунками.

Следует помнить, что решение каждой учебной задачи должно доводиться до окончательного логического ответа, которого требует условие, и по возможности с выводом. Полученный ответ следует проверить способами, вытекающими из существа данной задачи. Полезно также (если возможно) решать несколькими способами и сравнить полученные результаты. Решение задач данного типа нужно продолжать до приобретения твердых навыков в их решении.

2 Подготовка к практическим занятиям

Подготовку к каждому практическому занятию студент должен начать с ознакомления с методическими указаниями, которые включают содержание работы. Тщательное продумывание и изучение вопросов основывается на проработке теку-

щего материала лекции, а затем изучения обязательной и дополнительной литературы, рекомендованную к данной теме. На основе индивидуальных предпочтений студенту необходимо самостоятельно выбрать тему доклада по проблеме и по возможности подготовить по нему презентацию.

Если программой дисциплины предусмотрено выполнение практического задания, то его необходимо выполнить с учетом предложенной инструкции (устно или письменно). Все новые понятия по изучаемой теме необходимо выучить наизусть и внести в глоссарий, который целесообразно вести с самого начала изучения курса. Результат такой работы должен проявиться в способности студента свободно ответить на теоретические вопросы семинара, его выступлении и участии в коллективном обсуждении вопросов изучаемой темы, правильном выполнении практических заданий и контрольных работ.

В зависимости от содержания и количества отведенного времени на изучение каждой темы практическое занятие может состоять из четырех-пяти частей:

1. Обсуждение теоретических вопросов, определенных программой дисциплины.

2. Доклад и/ или выступление с презентациями по выбранной проблеме.

3. Обсуждение выступлений по теме – дискуссия.

4. Выполнение практического задания с последующим разбором полученных результатов или обсуждение практического задания.

5. Подведение итогов занятия.

Первая часть – обсуждение теоретических вопросов – проводится в виде фронтальной беседы со всей группой и включает выборочную проверку преподавателем теоретических знаний студентов. Примерная продолжительность — до 15 минут. Вторая часть — выступление студентов с докладами, которые должны сопровождаться презентациями с целью усиления наглядности восприятия, по одному из вопросов практического занятия. Обязательный элемент доклада – представление и анализ статистических данных, обоснование социальных последствий любого экономического факта, явления или процесса. Примерная продолжительность — 20-25 минут. После докладов следует их обсуждение – дискуссия. В ходе этого этапа практического занятия могут быть заданы уточняющие вопросы к докладчикам.

Примерная продолжительность – до 15-20 минут. Если программой предусмотрено выполнение практического задания в рамках конкретной темы, то преподавателями определяется его содержание и дается время на его выполнение, а замет идет обсуждение результатов. Подведением итогов заканчивается практическое занятие.

В процессе подготовки к практическим занятиям, студентам необходимо обратить особое внимание на самостоятельное изучение рекомендованной учебно-методической (а также научной и популярной) литературы. Самостоятельная работа с учебниками, учебными пособиями, научной, справочной и популярной литературой, материалами периодических изданий и Интернета, статистическими данными является наиболее эффективным методом получения знаний, позволяет значительно активизировать процесс овладения информацией, способствует более глубокому усвоению изучаемого материала, формирует у студентов свое отношение к конкретной проблеме. Более глубокому раскрытию вопросов способствует знакомство с дополнительной литературой, рекомендованной преподавателем по каждой теме семинарского или практического занятия, что позволяет студентам проявить свою индивидуальность в рамках выступления на данных занятиях, выявить широкий спектр мнений по изучаемой проблеме.

3 Самостоятельное изучение темы. Конспект

Конспект – наиболее совершенная и наиболее сложная форма записи. Слово «конспект» происходит от латинского «conspectus», что означает «обзор, изложение». В правильно составленном конспекте обычно выделено самое основное в изучаемом тексте, сосредоточено внимание на наиболее существенном, в кратких и четких формулировках обобщены важные теоретические положения.

Конспект представляет собой относительно подробное, последовательное изложение содержания прочитанного. На первых порах целесообразно в записях ближе держаться тексту, прибегая зачастую к прямому цитированию автора. В дальнейшем, по мере выработки навыков конспектирования, записи будут носить более свободный и сжатый характер.

Конспект книги обычно ведется в тетради. В самом начале конспекта указывается фамилия автора, полное название произведения, издательство, год и место из-

дания. При цитировании обязательная ссылка на страницу книги. Если цитата взята из собрания сочинений, то необходимо указать соответствующий том. Следует помнить, что четкая ссылка на источник – непременное правило конспектирования. Если конспектируется статья, то указывается, где и когда она была напечатана.

Конспект подразделяется на части в соответсвтии с заранее продуманным планом. Пункты плана записываются в тексте или на полях конспекта. Писать его рекомендуется четко и разборчиво, так как небрежная запись с течением времени становиться малопонятной для ее автора. Существует правило: конспект, составленный для себя, должен быть по возможности написан так, чтобы его легко прочитал и кто-либо другой.

Формы конспекта могут быть разными и зависят от его целевого назначения (изучение материала в целом или под определенным углом зрения, подготовка к докладу, выступлению на занятии и т.д.), а также от характера произведения (монография, статья, документ и т.п.). Если речь идет просто об изложении содержания работы, текст конспекта может быть сплошным, с выделением особо важных положений подчеркиванием или различными значками.

В случае, когда не ограничиваются переложением содержания, а фиксируют в конспекте и свои собственные суждения по данному вопросу или дополняют конспект соответствующими материалами их других источников, следует отводить место для такого рода записей. Рекомендуется разделить страницы тетради пополам по вертикали и в левой части вести конспект произведения, а в правой свои дополнительные записи, совмещая их по содержанию.

Конспектирование в большей мере, чем другие виды записей, помогает вырабатывать навыки правильного изложения в письменной форме важные теоретических и практических вопросов, умение четко их формулировать и ясно излагать своими словами.

Таким образом, составление конспекта требует вдумчивой работы, затраты времени и труда. Зато во время конспектирования приобретаются знания, создается фонд записей.

Конспект может быть текстуальным или тематическим. В текстуальном конспекте сохраняется логика и структура изучаемого произведения, а запись ведется в

соответствии с расположением материала в книге. За основу тематического конспекта берется не план произведения, а содержание какой-либо темы или проблемы.

Текстуальный конспект желательно начинать после того, как вся книга прочитана и продумана, но это, к сожалению, не всегда возможно. В первую очередь необходимо составить план произведения письменно или мысленно, поскольку в соответствии с этим планом строится дальнейшая работа. Конспект включает в себя тезисы, которые составляют его основу. Но, в отличие от тезисов, конспект содержит краткую запись не только выводов, но и доказательств, вплоть до фактического материала. Иначе говоря, конспект – это расширенные тезисы, дополненные рассуждениями и доказательствами, мыслями и соображениями составителя записи.

Как правило, конспект включает в себя и выписки, но в него могут войти отдельные места, цитируемые дословно, а также факты, примеры, цифры, таблицы и схемы, взятые из книги. Следует помнить, что работа над конспектом только тогда будет творческой, когда она не ограничена текстом изучаемого произведения. Нужно дополнять конспект данными из другими источников.

В конспекте необходимо выделять отдельные места текста в зависимости от их значимости. Можно пользоваться различными способами: подчеркиваниями, вопросительными и восклицательными знаками, репликами, краткими оценками, писать на полях своих конспектов слова: «важно», «очень важно», «верно», «характерно».

В конспект могут помещаться диаграммы, схемы, таблицы, которые придадут ему наглядность.

Составлению тематического конспекта предшествует тщательное изучение всей литературы, подобранной для раскрытия данной темы. Бывает, что какая-либо тема рассматривается в нескольких главах или в разных местах книги. А в конспекте весь материал, относящийся к теме, будет сосредоточен в одном месте. В плане конспекта рекомендуется делать пометки, к каким источникам (вплоть до страницы) придется обратиться для раскрытия вопросов. Тематический конспект составляется обычно для того, чтобы глубже изучить определенный вопрос, подготовиться к докладу, лекции или выступлению на семинарском занятии. Такой конспект по содер-

жанию приближается к реферату, докладу по избранной теме, особенно если включает и собственный вклад в изучение проблемы.

4 Комплект заданий для выполнения контрольной работы

4.1 Сокеты и протоколы

Под сокетом – абстрактное понятие, определяющее конечный пункт передачи данных, т.е. для того чтобы было установлено соединение между двумя программами необходимо, чтобы каждая из сетевых программ имела свой собственный сокет.

Сокеты Windows основываются на следующих элементах:

- Wsock32.dll – компонент, который разделяет адресное пространство с приложением, работающим в режиме пользователя и использующим сокеты Windows.
- Эмулятор сокетов Windows, который обеспечивает трансляцию вызовов от приложений, использующих сокеты Windows, в вызовы для протоколов, совместимых с TDI (Transport driver interface).

Сокет имеет следующую структуру:

1. Семейство протоколов
2. Тип сервиса
3. Локальный IP-адрес
4. Удаленный IP-адрес
5. Локальный порт протокола
6. Удаленный порт протокола

В настоящее время протоколами, поддерживающим сокеты Windows являются: TCP/IP и NWLink (IPX/SPX). Для работы по созданию, настройке и поддержанию в работе используется специальный набор API, представляющий собой интерфейс прикладного программирования.

Приложение использующее механизм сокетов

Рисунок 1.1 – Программный интерфейс сокетов Windows

Интерфейс прикладного программирования сокетов в ОС Windows содержит три вида функций:

- функции сокетов в стиле Беркли:

Функция	Описание
accept() *	Переводит сокет в состояние ожидания. Созданное с внешним сокетом соединение подтверждается
bind()	Связывет локальное имя с созданным сокетом
closesocket() *	Удаляет сокет и все связанные сним процессы
connect() *	Производит ориентированное на соединение подключение

Функция	Описание
getpeername()	Возвращает имя сокета, связанного с заданным сокетом.
getsockname()	Возвращает адрес сокета, связанного с заданным сокетом.
getsockopt()	Возвращает опции сокета
htonl() ∞	Конвертирует 32-битное число из формата хоста в формат сети.
htons() ∞	Конвертирует 16-битное число из формата хоста в формат сети..
ntohl() ∞	Конвертирует 32-битное число из формата сети в формат. хоста
ntohs() ∞	Конвертирует 16-битное число из формата сети в формат. хоста
inet_addr() ∞	Конвертирует строковую переменную в стандартную переменную адреса ИНТЕРНЕТ
inet_ntoa() ∞	Конвертирует стандартную переменную адреса ИНТЕРНЕТ в строковую переменную
ioctlsocket()	Обеспечивает контроль сокета
listen()	Прослушивает соединение созданного сокета
recv() *	Принимает данные из ориентированного на соединение сокета
recvfrom() *	Принимает данные из неориентированного на соединение сокета
select() *	Определение состояния сокета
send() *	Посылает данные в соединенный сокет
sendto() *	Посылает данные как в соединенный та не соединенный сокет
setsockopt()	Устанавливает опции сокета
shutdown()	Закрывет часть дуплексного соединения
socket()	Создает сокет и возвращает дескриптор сокета.

- функции для работы с базами данных (функции типа **getXbyY**()):

Функция	Описание
Gethostbyaddr	возвращает указатель на структуру хост-информации
Gethostbyname	Известно IP-имя, то вызывается для определения IP-адреса
Gethostname	Возвращает имя локальной ПЭВМ
Getprotobyname	Возвращает официальное имя и номер протокола по указанному имени (например TCP)
Getprotobynumber	Возвращает имя и номер протокола по указанному номеру

Функция	Описание
Getservbyname	Возвращает имя сетевой службы и номер порта протокола, соответствующие указанному имени
Getprotobyport	Возвращает имя сетевой службы и номер порта протокола, соответствующие указанному номеру порта –

- функции характерные только для WINDOWS.

Рассмотрим описание некоторых функций.

struct hostent FAR * WSAAPI

gethostbyaddr (IN const char FAR * *addr*, **INint** *len*, **IN int** *type* **);**

addr – указатель на адрес в формате сети.

len – длина адреса.

type – тип адреса.

gethostbyaddr() возвращает указатель на следующую структуру

```
struct hostent {
        char FAR *        h_name;
        char FAR * FAR *     h_aliases;
        short h_addrtype;
        short h_length;
        char FAR * FAR *     h_addr_list;
};
```

Элементы структуры следующие:

h_name – официальное имя хоста (PC);

h_aliases – NULL-terminated массив альтернативных имен;

h_addrtype – тип возвращаемых адресов;

h_length – длина в байтах каждого адреса;

h_addr_list – NULL-terminated список адресов в формате сети;

struct hostent FAR * WSAAPI

gethostbyname (IN const char FAR * *name***);**

name – указатель null-terminated на имя хоста.

gethostname (OUT char FAR * *name*, IN int *namelen*);

name – указатель на буфер,в котором должно находиться имя машины.

namelen – длина буфера.

struct protoent FAR * WSAAPI

getprotobyname (IN const char FAR * *name*);

name – указатель null terminated на структуру протокола.

```
struct protoent {
        char FAR *  p_name;
        char FAR * FAR *        p_aliases;
        short  p_proto;
};
```

Состав структуры следующий:

p_name – имя протокола.

p_aliases – NULL-terminated массив других имен.

p_proto – номер протокола в сетевом формате.

struct protoent FAR * WSAAPI

getprotobynumber (IN int *number*);

number – номер протокола в сетевом формате.

struct servent FAR * WSAAPI

getservbyname (IN const char FAR* *name*, IN const char FAR * *proto*);

name – указатель a null terminated на имя службы.

proto – указатель null terminated на имя протокола.

```
struct servent {
        char FAR *  s_name;
        char FAR * FAR *        s_aliases;
        short  s_port;
        char FAR *  s_proto;
};
```

Состав структуры следующий:

s_name – имя службы.

s_aliases – NULL-terminated массив других имен.

s_port – номер порта с которым служба может контактировать. Порт возвращается в формате сети.

s_proto – имя протокола контактирующего со службой.

struct servent FAR * WSAAPI

getservbyport (IN int *port*, **IN const char FAR*** *proto* **);**

port – порт для службы в сетевом порядке.

proto – указатель на имя протокола

SOCKET WSAAPI socket (IN int ***af***, **IN int** ***type***, **INint** ***protocol*** **);**

af – семейство протоколов.

type – тип сокета (ориентирован на соединение SOCK_STREAM или SOCK_DGRAM).

protocol – один из протоколов семейства, используемый для соединения

int WSAAPI sendto (IN SOCKET *s*, **IN const char FAR*** *buf*, **IN int** *len*, **IN int** *flags*, **IN const struct sockaddr FAR *** *to*, **IN int** *tolen* **);**

s – дескриптор сокета;

buf – буфер данных для передачи;

len – длина буфера;

flags – опции передачи (MSG_DONTROUTE – без маршрутизации MSG_OOB – срочная передача и т .д);

to – адрес, принимающий сокет;

tolen – размер адреса принимающего сокет.

После каждой вызванной функции целесообразно проверять наличие ошибок. Для чего необходимо сравнить дескриптор со значением INVALID_SOCKET, а во всех других случаях использовать функцию **WSAGetLastError().**

4.2 Процесс установления соединения

При вызове функции *soket* из таблицы дескрипторов процесса выбирается указатель на внутреннюю структуру данных сокета. Известно, что для передачи ин-

формации от одного прикладного процесса к другому пользуясь стеком протоколов TCP/IP необходимо указать: порт, через который прикладная программа свяжется с транспортным протоколом; протоколы, используемые прикладная программа (TCP, UDP, IP, ICMP); в заголовке IP-адреса необходимо, кому и от кого передается информация; из какого порта прикладной процесс-получатель получит информацию.

Заполненная полностью структура данных сокета обладает полной информацией для ОС, необходимой для организации связи.

Очевидно, что при вызове функции *soket* из данного дескриптора определяются только:

1. *Protocol_fam* – семейство протоколов;
2. *Protocol* – тип сервиса

Другие параметры не определены. Необходимо определить номера портов и IP-адреса. Настройкой сокета называется процесс заполнения дескриптора созданного ранее сокета. Данная настройка осуществляется специальными API-функциями, так как нужно сетевой программе в зависимости от того, что от нее требуется.

Применение данных функций характерно, как для UNIX BSD, так и для сокетов Windows.

<table>
<tr><th>Использование сокета</th><th>Местный процесс</th><th>Удаленный процесс</th></tr>
<tr><td>Ориентированный на соединение клиент</td><td colspan="2">Вызов функции CONNECT( ) записывает в структуру данных сокета информацию как о местном так и об удаленном участках соединения. Местный IP-адрес и порты ОС разместит самостоятельно.</td></tr>
<tr><td>Ориентированный на соединение сервер</td><td>BIND( )</td><td>LISTEN( )
ACCEPT( )</td></tr>
<tr><td>Не ориентированный на соедине-</td><td>BIND( )</td><td>SENDTO( )</td></tr>
</table>

ние клиент		
Не ориентированный на соединение сервер	*BIND( )*	*RECVFROM( )*

Например, *connect(sock_handle, rem_sock_addr, addr_length)*. Третий параметр сообщает длину структуры данных адреса. Длина этого адреса для разных сетей может быть различным.

В случае необходимости передачи информации потоком датаграмм необходимо реализовать режим прослушивания порта обмена. Например, сервер как ориентированный на соединение, так не ориентированный должны постоянно ожидать вызова клиента. Это осуществляется следующей функцией: *bind(sock_handle, loc_sock_addr, addr_length);*

При этом, регистрируется собственный номер порта и данному порту назначается приложение. Другими словами, программа говорит, что она ожидает данные из порта. После вызова указанных функций сокет считается сконфигурированным и необходимо наладить соединение, т.е. осуществить пересылку информации. Для этого используются 5 основных функций передачи и приема:

N	Сокеты Unix BSD		Сокеты 1.1 Windows		Описание
	Функция передачи	Функция приема	Функция передачи	Функция приема	
1	*send*	*recv*	*send*	*recv*	Передает (принимает) данные через соединенный сокет. Использует некоторые флаги для управления поведением сокета. Не требует задания адреса назначения
2	*write*	*read*	—	—	Передает (принимает) данные через соединенный сокет. Для передачи используется буфер данных. Не требует задания адреса назначения
3	*writev*	*readv*	—	—	Передает (принимает)данные через соединенный сокет. В качестве буфера используются раздельно расположенные блоки памяти.

4	*sendto*	*recvfrom*	*sendto*	*recvfrom*	Передает данные через не соединенный сокет. Для передачи используется буфер данных. Необходимо указывать адрес назначения
5	*sendmsg*	*recvmsg*	—	—	Передает (принимает) данные через не соединенный сокет. В качестве буфера используется гибкая структура сообщения. Необходимо указывать адрес назначения

send(sock_handle, message_buffer, buffer_length, special_flags),

где *special_flags* – позволяет задавать флаги для управления передачей данных (например, приоритета). Обычно, если нет дополнительных условий как на приеме так и при передаче значение *flags=0;*

sendto(sock_handle, message_buffer, buffer_length, special_flags, sock_addr_struct, addr_struct_length);

recfrom(sock_handle, message_buffer, buffer_length, special_flags, sock_addr_struct, addr_struct_length),

где sock_addr_struct – адрес назначения датаграммы с определенной длиной – addr_struct_length;

Для принимающих функций синтаксис аналогичен. Однако не обязательно строгое соответствие вызывающих передачу и принимающих функций. Например, передача через *send* может приниматься как *recv, read, readv*. Рассмотрим ситуацию, когда на вход нашего сервера поступает несколько вызовов одновременно. Для эффективного разрешения конфликтных ситуаций используется функция *listen* – переводит сокет в состояние прослушивания и предназначена для установления максимального ограничения очереди. Например, *listen(sock_handle, queue_length)*

Функция *accept* позволяет принять запрос на соединение, поступивший от клиента. После того, как установлена входная очередь, программа-сервер вызывает функцию *accept* и переходит в режим ожидания (паузы).

accept(sock_handle, sock_addr, addr_length);

Как только появится запрос от клиента, второй параметр получит значение адреса клиента и длину адреса. Другими словами, как только на сокете, контролируемом функцией *accept* – появится очередной запрос клиента, то программное обеспечение сервера создает новый сокет и автоматически, зная адрес клиента производит соединение с запрашивающим соединение. Выходным аргументом функции является сокет. Таким образом, прием информации будет осуществляться не первичным сокетом, образованным командой *soket,* а вторичным сокетом, на который указывает *socket* от *accept.* Следовательно, при использовании функций приема и передачи для сервера необходимо в качестве аргумента использовать *socket* от *accept.* Общие схемы установления соединений представлены на рисунке 1.2 и рисунке 1.3.

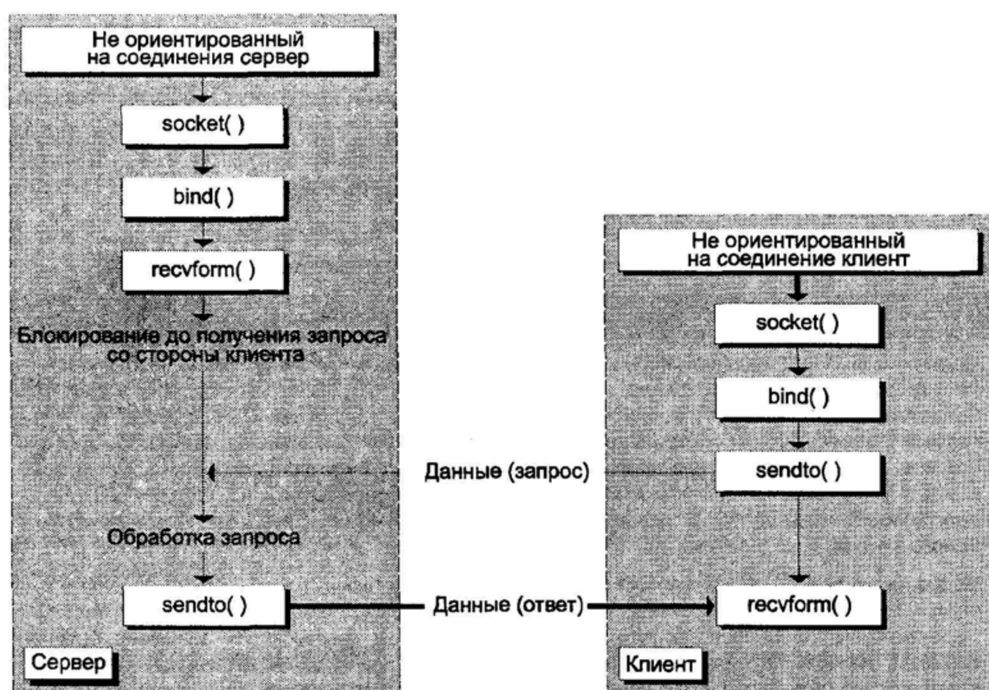


Рисунок 1.2 – Сокет с UDP протоколом

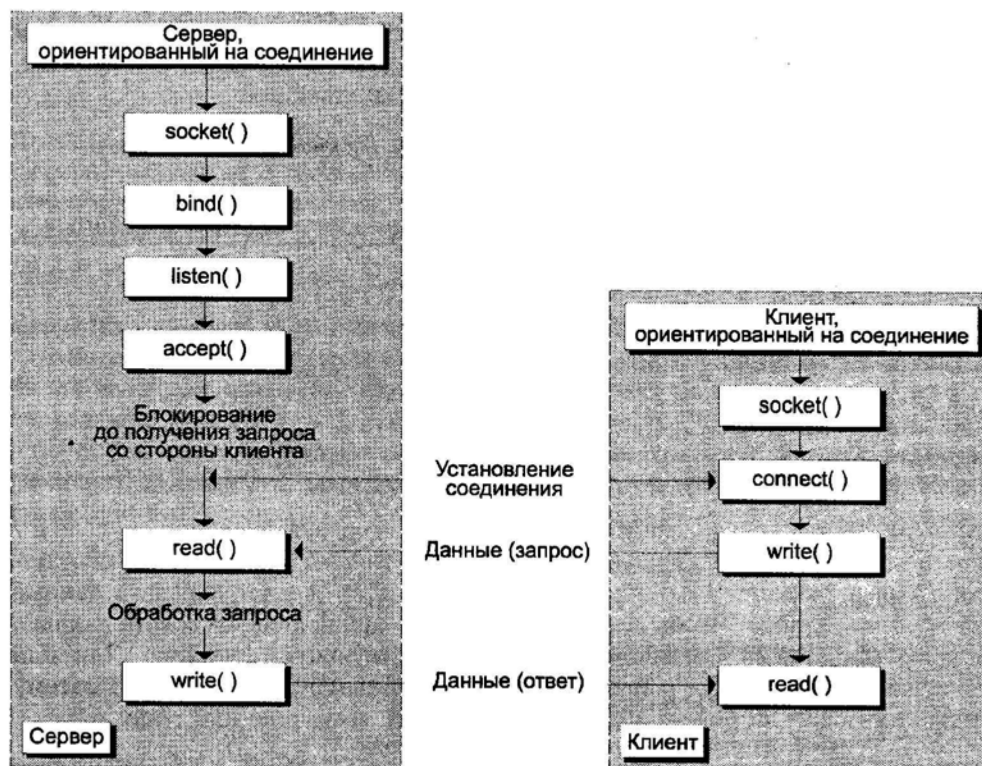


Рисунок 1.3 – Сокет с TCP протоколом

Для определения состояния одновременно нескольких сокетов используется функция *select*. Напрмер, *select(number_of_sock, read_sock, write_sock, error_sock, max_time),* где *read_sock, write_sock, error_sock* – битовые маски задающие тип сокетов.

В лабораторной работе целесообразно использовать функцию WSAASyncSelect, которая позволяет оживить интерфейс за счет неблокирующего вызова.

4.3 Порядок выполнения контрольной работы

1. Изучить принципы сетевого взаимодействия Windows-приложений.
2. Разработать сетевые оконные приложения.
3. Защитить лабораторную работу преподавателю.

Таблица 5.1 – Варианты индивидуальных заданий на контрольную работу

Вариант	Задание на контрольную работу			
	транспортный протокол	IP-адрес	номер порта	данные
1	UDP	в коде	из Edit	из Edit
2	TCP	в коде	из Edit	из файла
3	UDP	из Edit	в коде	из Edit

4	TCP	из Edit	в коде	из файла
5	UDP	в коде	в коде	из Edit
6	TCP	в коде	в коде	из файла
7	UDP	из Edit	из Edit	из Edit
8	TCP	из Edit	из Edit	из файла
9	UDP	из файла	в коде	из Edit
10	TCP	из файла	в коде	из файла
11	UDP	в коде	из файла	из Edit
12	TCP	в коде	из файла	из файла
13	UDP	из Edit	из файла	из Edit
14	TCP	из Edit	из файла	из файла
15	UDP	из файла	из файла	из Edit
16	TCP	из файла	из файла	из файла
17	UDP	из файла	из Edit	из Edit
18	TCP	из файла	из Edit	из файла

Контрольные вопросы

1. Что такое сокет? Какие типы сокетов Вам известны?

2. Какие API функции используются для установления сеанса в соответствии с протоколом TCP?

3. Как реализовать широковещательную рассылку пакетов в компьютерной сети?

4. Чем отличаются сокеты Windows от сокетов Unix?

5. Состояния TCP-соединения.

6. Какие функции интерфейса прикладного программирования сокетов в ОС Windows Вам известны?

Литература

Основная литература

1. Новиков Ю.В. Основы микропроцессорной техники. — Электрон. текст. дан.— М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — Режим доступа : [http ://www.iprbookshop.ru/52207](http://www.iprbookshop.ru/52207). — ЭБС «IPRbooks», по паролю.

2. Водовозов А.М. Микроконтроллеры для систем автоматики : Учеб. пособие. — Электрон. текст. дан. — М. : Инфра-Инженерия, 2016. — Режим доступа : [http ://www.iprbookshop.ru/51727](http://www.iprbookshop.ru/51727).— ЭБС «IPRbooks», по паролю

Дополнительная литература

1. Гуров В.В. Архитектура микропроцессоров [Электронный ресурс]/ Гуров В.В.— Электрон. текстовые данные.— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 115 c.— Режим доступа: http://www.iprbookshop.ru/56313.html.— ЭБС «IPRbooks»

2. Аблязов Р.З. Программирование на ассемблере на платформе x86-64 [Электронный ресурс]/ Аблязов Р.З.— Электрон. текстовые данные.— Саратов: Профобразование, 2017.— 304 c.— Режим доступа: http://www.iprbookshop.ru/63951.html.— ЭБС «IPRbooks»