ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра информационных систем и технологий

О.Л. Куляс, К.А. Никитин

Аппаратные интерфейсы ЭВМ

Лабораторный практикум
по дисциплине «ЭВМ и периферийные
устройства»
(часть 3)

Самара
2017

УДК 004.43 (076)

К 907

Куляс, О.Л.

К 907 Аппаратные интерфейсы ЭВМ: лабораторный практикум по дисциплине «ЭВМ и периферийные устройства» (часть 3) / О.Л. Куляс, К.А. Никитин. – Самара: ПГУТИ, 2017. – 100 с.

Лабораторный практикум предназначен для бакалавров направления 09.03.01 – «Информатика и вычислительная техника», изучающих курс «ЭВМ и периферийные устройства». Двухсеместровый цикл лабораторных работ включает 17 работ (7 работ в 1-й части, 5 работ во 2-й и 5 работ в 3-й). Каждая лабораторная работа содержит достаточный теоретический материал, поэтапно вводящий студентов в мир программирования на языке Ассемблера, сведения и задания, необходимые для практического выполнения работы, список литературы, рекомендуемой для дополнительного изучения, а также контрольные вопросы для проверки усвоения изученного.

Содержание

Лабораторная работа №1
Инструментарии операционной системы для сбора информации о компонентах ПЭВМ

1. Цель работы

Практическое ознакомление со структурой ПЭВМ, методами определения ее конфигурации, ресурсами, выделяемые компонентам ПЭВМ.

2. Теоретический материал

Персональные ЭВМ имеют модульную структуру для обеспечения создания из базовых модулей необходимую для пользователя конфигурацию машины. Обычно в любой ПЭВМ имеются следующие узлы:

- процессор;
- материнская (системная) плата;
- оперативная память;
- видеоадаптер;
- жесткий диск;
- корпус системного блока с блоком питания:
- монитор;
- клавиатура;
- мышь.

Определить конфигурацию уже собранного ПЭВМ можно различными способами:

- визуально осмотреть компоненты ПЭВМ на предмет наличия маркировок производителей;
- опросить компоненты ПЭВМ программным способом;
- просмотреть информацию в BIOS;
- воспользоваться служебными программами операционной системы («Сведения о системе» MSinfo32.exe, «Диспетчер устройств» Devmgmt.msc, «Средство диагностики DirectX» Dxdiag.exe, «Консольная утилита для вызова объектов и методов WMI» Wmic.exe, «Средство оценки производительности» WinSat.exe и др.);
- запустить специализированные диагностические и тестовые программы (AIDA64 от FinalWire Ltd, Sandra от SiSoftware, Speccy от Piriform и др.).

В операционных системах семейства Windows большая часть информации об аппаратном обеспечении системы хранится в реестре, однако работать с ней не очень удобно, поэтому для её визуального представления используются специальные утилиты. Основным источником информации об аппаратной части ПЭВМ может служить утили-

та *«Диспетчер устройств»*. Данная утилита в древовидном виде представляет информацию о компонентах системы с возможностью просмотра детальной информации с вызовом через контекстное меню. В ОС Windows Vista/7/8 «Диспетчер устройств» можно найти в меню Пуск / Панель управления / Оборудование и звук. Диспетчер устройств можно также открыть, набрав в поисковой строке меню Пуск или в приложении Выполнить (открывается нажатием комбинацией клавиш **Win**+**R**)

Devmgmt.msc.

Для просмотра информации об аппаратной и программной частях системы можно воспользоваться утилитой «*Сведения о системе*». Для этого наберите в поисковой строке меню Пуск или в приложении Выполнить

MSinfo32.exe.

Системная информация на панели обзора данной утилиты разделена на четыре категорий: корневой узел *Сведения о системе* и три основных подузла:

- аппаратные ресурсы;
- компоненты;
- программная среда.

Основную информацию о видеоадаптере и звуковой карте можно получить через утилиту «*Средство диагностики DirectX»*. Для этого необходимо в поисковой строке меню Пуск или в приложении Выполнить набрать

Dxdiag.exe.

Запрос проверки цифровых подписей драйверов можно игнорировать.

2.1. Программное получение информации о процессоре

Для программной идентификации большинство современных процессоров поддерживает команду **CPUID**. Данная команда впервые появилась в процессоре Pentium и предоставляет программному обеспечению информацию о производителе, семействе, модели и поколении процессора, наборе поддерживаемых процессором функций и другую информацию.

Команда CPUID не имеет операндов, однако использует регистр EAX для указания номера функции (входное значение). Для разных моделей процессоров задокументированы различные наборы допустимых входных значений EAX. В общем случае, их можно разделить на

стандартные (поддерживаемые всеми производителями) и расширенные (так или иначе отличающиеся для процессоров разных моделей и производителей).

Вся информация, даже текстовая, возвращается в регистрах EAX, EBX, ECX и EDX. В зависимости от запрашиваемой информации (входного значения в EAX) назначение регистров будет разным.

Если входное значение ***EAX=0***, то после выполнения команды CPUID регистр EAX будет содержать максимальное значение, понимаемое командой CPUID (только стандартные функции), а в регистрах EBX, EDX и ECX будет находится строка идентификации производителя процессора (EBX содержит первых 4 символа, EDX содержит следующие 4 символа и ECX содержит последние 4 символа). В табл. 1.1 приведены значения, выдаваемые наиболее распространенными моделями микропроцессоров (МП).

Таблица 1.1

Содержимое регистров после выполнения команды CPUID (EAX = 0)

Производитель	**EAX**	**EBX:EDX:ECX**	
		ASCII-строка	**HEX-значения**
Intel	x	GenuineIntel	756E6547:49656E69:6C65746E
AMD	1	AuthenticAMD	68747541:69746E65:444D4163
Cyrix	1	CyrixInstead	69727943:736E4978:64616574
Centaur	1	CentaurHauls	746E6543:48727561:736C7561
SiS	1	SiS SiS SiS	20536953:20536953:20536953

После выполнения команды CPUID с входным значением ***EAX=1***, биты регистра EAX будут иметь следующие назначения (т.н. сигнатура процессора):

EAX[3:0] – (Stepping) номер разработки МП;

EAX[7:4] – (Base Model) модель МП,

EAX[11:8] – (Base Family) номер семейства МП;

EAX[13:12] – (Type) тип МП (00b – стандартный, 01b – OverDrive, 10b – Dual, 11b –зарезервировано; процессоры Pentium OverDrive for Pentium возвращают 00b в поле тип);

EAX[15:14] – зарезервировано;

EAX[19:16] – (Extended Model) расширение поля модели МП;

EAX[27:20] – (Extended Family) расширение поля семейства МП;

EAX[31:28] – зарезервировано.

Семейство МП (Family) объединяет несколько выпускаемых процессоров в одну группу, обладающую некоторыми общими свойствами программного и аппаратного обеспечения. Модель (Model) вы-

деляет один экземпляр из семейства МП. Номер разработки (Stepping) определяет конкретную версию определённой модели.

Семейство МП определяется как сумма чисел в Base Family и Extended Family, однако если значение Base Family меньше Fh, то Extended Family не используется, а в качестве семейства МП используется только поле Base Family. Например, если Base Family = Fh, а Extended Family = 1h, то семейство МП будет 10h.

Модель МП формируется объединением двух чисел в одной записи слева на право: сначала записывается Extended Model, а затем Base Model. Если Base Model меньше чем Fh, то Extended Model не используется, а модель МП формируется только из Base Model. Рассмотрим пример: Base Model = 8h, Extended Model = Eh, тогда модель МП будет E8h. Программно реализовать объединение двух полей можно выделив из сигнатуры соответствующие поля в отдельные регистры, выполнить смещение на 4 разряда влево поля Extended Model с заполнением нулями справа и сложить содержимое используемых регистров.

Регистры EBX и ECX при входном значении EAX=1 возвращают информацию о характеристиках процессора.

В регистре EDX будет содержаться информация о некоторых свойствах и возможностях микропроцессора, например, о поддержке команд 3DNow!, MMX, SIMD. Установленный флаг свойств указывает на то, что соответствующее свойство (возможность, функция) данной моделью микропроцессора поддерживается. Назначение битов регистра EDX можно найти в рекомендованной литературе [7.4].

Команда CPUID с входным значением ***EAX = 2*** предназначена для получения информации об объеме и типе КЭШ памяти микропроцессора. После выполнения этой команды в регистрах EAX, EBX, EDX, ECX содержится соответствующая информация, причем, в младших 8-ми битах регистра EAX (регистр AL) содержится число, указывающее на то, сколько раз подряд необходимо выполнить команду CPUID со входным EAX = 2, чтобы получить достоверную информацию о микропроцессоре (в случае AL = 1, команда должна выполняться однократно).

Команда CPUID с входным значением ***EAX = 3*** предназначена для чтения т.н. идентификационного номера микропроцессора – уникального 96-разрядного номера. После выполнения этой команды регистр ECX содержит младшие 32 бита идентификационного номера, а регистр EDX – средние 32 бита. Старшие 32 бита идентификационного номера – это данные, выдаваемые в регистре EAX командой CPUID с

входным EDX =1. Микропроцессоры, начиная с Pentium 4, не выводят идентификационные номера.

Для некоторых процессоров-клонов (AMD, Cyrix, IDT) определены т.н. нестандартные (расширенные) функции команды CPUID. Они вызываются при входных значениях в EAX больших 7FFFFFFFh. Полное описание этих функций вы сможете найти в технических руководствах фирм-производителей по конкретным процессорам, а здесь приведем только функцию для получения полного наименования процессора.

Команда CPUID с входными значениями в ***EAX=80000002h***, ***80000003h***, ***80000004h*** возвращает последовательно в регистрах EAX, EBX, ECX, EDX 48-ми значную ASCII строку, содержащую полное наименование процессора. Если строка короче 48 символов, то она будет дополняться пробелами. Эта строка обычно используется BIOS для вывода на экран ПК при начальной загрузке.

Перед использованием команды CPUID необходимо убедиться, что данная команда поддерживается процессором. Для этого необходимо попытаться изменить 21-й бит (флаг идентификации ID) расширенного регистра EFLAGS. Если бит успешно поменяется, то инструкция CPUID процессором поддерживается, если регистр флагов остался без изменений, то процессор команду CPUID не поддерживает.

Если проверка прошла успешно, необходимо определить максимальное количество поддерживаемых стандартных функций команды CPUID и производителя микропроцессора.

Пример 32-х разрядного приложения на Ассемблере, выводящего в консоль информацию о количестве поддерживаемых функций команды CPUID процессором приведен ниже.

2.2. Создание 32-х разрядных приложений на Ассемблере

Операционная система Windows в зависимости от версии позволяет запускать 16-ти, 32-х и 64-х разрядные приложения, однако 16-ти разрядные приложения поддерживаются только 32-х разрядными версиями Windows. Этот момент необходимо учитывать при создании приложения на языке Ассемблера. Современные пакеты языка Ассемблер позволяют создавать как 16-ти разрядные, так и 32-х и 64-х разрядные приложения. В дальнейшем будет рассмотрено создание 32-х разрядного приложения с использованием пакета Ассемблера **Turbo Assembler** версии 5.3 от компании Borland.

Отличительной особенностью создания 32-х разрядных приложений под Windows является отсутствие доступа к портам ввода-вывода и программным прерываниям BIOS и DOS. Взамен операцион-

ная система предоставляет свои программные средства, которые называются **Win API**. Win API включает в себя набор функций, находящихся в библиотеках kernel32.dll, user32.dll, gdi32.dll, advapi32.dll и др. Для того чтобы использовать некоторую функцию этих библиотек необходимо загрузить нужную библиотеку в свой исходный модуль (например, директивой ***INCLUDELIB***) и указать какие функции будут внешними с помощью ***EXTRN***.

Вызов API функции осуществляется командой **call**, а параметры функции передаются заранее через стек. Порядок передачи параметров в функцию определяется моделью вызова, которая указывается после задания модели памяти. В случае использования модели ***STDCALL*** параметры передаются слева направо и снизу вверх. Например, вызов функции с параметрами

функция (параметр 1, параметр 2, параметр 3)

на языке Ассемблера будет выглядеть следующим образом

```
PUSH параметр 3
PUSH параметр 2
PUSH параметр 1
CALL функция
```

Однако допустимо указывать передаваемые параметры в самой команде **call**:

```
CALL функция, параметр 1, параметр 2, параметр 3
```

Для создания самого простого консольного приложения понадобятся три функции из библиотеки *Kernel32.lib*, поэтому далее рассмотрим их подробнее.

Приложение в ОС Windows может создавать собственные окна или использовать уже существующее, если необходимо вывести информацию в существующую консоль. Для получения дескриптора существующего консольного окна можно использовать функцию ***GetStdHandle***, в качестве аргумента которой указывается одна из трех констант

```
STD_INPUT_HANDLE    equ –10     ;для ввода
STD_OUTPUT_HANDLE   equ –11     ;для вывода
STD_ERROR_HANDLE    equ –12     ;для сообщения об ошибке
```

Функция возвращает дескриптор в регистр EAX.

Для вывода текстовой информации в консоль используется API-функция ***WriteConsole***, общий формат которой имеет следующий вид:

WriteConsole *(hConsoleOutput, lpBuffer, nNumberOfCharsToWrite, lpNumberOfCharsWritten, lpReserved),*

где *hConsoleOutput* – дескриптор буфера вывода консоли;

lpBuffer – указатель на буфер, где находится выводимый текст;

nNumberOfCharsToWrite – количество выводимых символов;

lpNumberOfCharsWritten – указывает на переменную, куда будет помещено количество действительно выведенных символов;

lpReserved – резервный параметр, должен быть равен нулю.

Все функции Win API, которые работают со строковыми данными, могут обрабатывать как ANSI кодировку, так и UNICODE. Добавление к имени функции суффикс «A» означает, что строковые данные должны иметь кодировку в стандарте ANSI. В дальнейшем будем использовать именно эту кодировку.

Каждая программа в конце своего выполнения обязательно должна вызвать функцию ***ExitProcess*** для удаления приложения из памяти.

В 32-х разрядных приложениях Windows используется так называемая плоская (**flat**) модель памяти, в которой базовые линейные адреса сегментов команд и данных равны 0, а размер достигает 4 Гбайт.

Трансляция исходного модуля производится программой **tasm32.exe**, общий формат вызова которой из командной строки выглядит следующим образом:

tasm32 </ключи> <имя файла>.

В качестве ключа рекомендуется использовать **/mx**, чтобы регистр символов в названиях внешних процедур соответствовал одноименным процедурам во внешнем файле. Информацию о других возможных ключах программы **tasm32.exe** можно получить, вызвав справку командой **tasm32 /?**

Компоновка объектных модулей производится вызовом компоновщика **tlink32.exe** из командной строки

tlink32 /Tpe /ap /v <имя файла>,

где **/Tpe** – ключ создания 32-х разрядного exe-приложения;

/ap – ключ создания 32-х разрядного консольного приложения;

/v – ключ, передающий в загрузочный файл информацию, используемую при отладке программ.

```
TITLE CPUID
;32-х разрядное приложение получения информации о процессоре,
;используя команду CPUID.
;Программа выводит в текущую консоль количество поддерживаемых
;процессором стандартных функций команды CPUID.
.586                                ;Расширенная система команд.
;Плоская модель памяти и стандартная модель вызова подпрограмм.
.MODEL FLAT, STDCALL
;Директивы компоновщику для подключения библиотек.
INCLUDELIB import32.lib             ;Работа с библиотекой ОС Kernel32.

;Внешние процедуры
EXTRN GetStdHandle:  PROC           ;Получить дескриптор окна.
EXTRN WriteConsoleA: PROC           ;Вывести в консоль.
EXTRN ExitProcess:   PROC           ;Завершить процесс.

;Константы
Std_Output_Handle EQU -11           ;Консольное окно для вывода данных.

.DATA                               ;Открыть сегмент данных.
  Handl_Out DD ?                    ;Дескриптор окна вывода данных.
  Lens DW ?                         ;Количество выведенных символов.
  Not_Supp DB 'CPUID not supported', 13, 10, '$'
  Not_Supp_L = $ - Not_Supp - 1     ;Длина строки Not_Supp без знака $.
  Supp DB 'CPUID supported', 13, 10, '$'
  Supp_L = $ - Supp - 1
  Str1 DB 'Number function: ','$'
  Str1_L = $ - Str1 - 1
  Number DD ?
  Number_Str DB 8 dup (0)
  Number_Str_L = 8
;-------------------------------------------------------------------------------------
.CODE                               ;Открыть сегмент кодов.
  Preobr PROC
    mov EAX, Number
    mov EBX, 10
    mov word ptr Number_Str, 0      ;Очистить переменную.
    mov word ptr Number_Str[2], 0h
    mov word ptr Number_Str[4], 0h
    mov word ptr Number_Str[6], 0h
    xor ECX, ECX
```

```
  m1: xor EDX, EDX
    div EBX
    push EDX
    inc ECX
    cmp EAX, 0
    jne m1
    xor ESI, ESI
  m2: pop EAX
    add EAX, 30h
    mov Number_Str[ESI], AL
    inc ESI
    loop m2
    ret
  Preobr ENDP

  Start:
;Получить дескриптор окна вывода.
    call GetStdHandle, Std_Output_Handle
    mov Handl_Out, EAX
  ;--------Проверка поддержки команды CPUID процессором----------
    pushfd              ;Получение регистра флагов через стек.
    pop  EAX
    mov  EBX, EAX
    xor  EAX, 200000h   ;Изменение 21-ого бита в регистре флагов.
    push  EAX           ;Сохранение и снова получение
    popfd               ;регистра флагов.
    pushfd
    pop  EAX
    cmp  EAX, EBX       ;Сравнение регистров до и после изменения.
    jne CPUIDSupp       ;Если не изменился, то CPUID не поддерживается.
;Вывести строку Not_supp.
    call WriteConsoleA, Handl_Out, offset Not_Supp, Not_Supp_L, offset Lens, 0
    jmp @exit
  CPUIDSupp:                    ;Вывести строку Supp.
    call WriteConsoleA, Handl_Out, offset Supp, Supp_L, offset Lens, 0

;Получение информации о количестве поддерживаемых функций CPUID
    mov EAX, 0
    cpuid
    mov Number, EAX             ;Количество поддерживаемых функций.
    call Preobr
```

```
;Добавление в конец строки служебных кодов 13 и 10.
    mov word ptr Number_Str[6], 0D0Ah
;Вывести строку Str1.
    call WriteConsoleA, Handl_Out, offset Str1, Str1_L, offset Lens, 0
;Вывести строку Number_Str.
   call WriteConsoleA, Handl_Out, offset Number_Str, Number_Str_L, offset Lens, 0
@exit: call ExitProcess                ;Завершить программу.
END Start                              ;Конец исходного модуля.
```

2.3. Средство администрирования WMI и утилита WMIC

Среди инструментов и средств автоматизации в операционной системе Windows особое место занимает технология **Windows Management Instrumentation** (**WMI**). Технология WMI – это созданная фирмой Microsoft реализация модели управления предприятием на базе Web (Web-Based Enterprise Management, WBEM), которая разработана и принята рабочей группой по управлению распределенными системами (Distributed Management Task Force, DMTF), при участии таких компаний, как BMC Software, Cisco Systems, Intel и Microsoft. Задачей WBEM была разработка таких стандартов для удаленного управления информационной средой предприятия, которые позволили бы управлять всеми физическими и логическими компонентами этой среды из одной точки и не зависели бы при этом от конкретного оборудования, сетевой инфраструктуры, операционной системы, файловой системы и т. д. Для этого была предложена модель CIM (Common Information Model), которая представляет физическую и логическую структуры компьютерной системы в виде единой расширяемой объектно-ориентированной информационной модели и определяет единые интерфейсы для получения информации о любом компоненте этой модели.

Технология WMI – это глобальная концепция настройки, управления и слежения за работой различных частей корпоративной компьютерной сети под управлением платформы Windows.

Для доступа к WMI можно использовать различные механизмы. Приложения могут обращаться к WMI через WMI COM API. Web-приложения могут использовать средства управления ActiveX для создания сетевых интерфейсов к данным WMI. Системные администраторы могут создавать сценарии, выполняемые в среде Windows Script Host (WSH), или формировать запросы через консольную утилиту WMIC. В данной работе для доступа к средствам WMI будет использоваться консольная утилита WMIC (WMI command-line), которая присутствует в операционной системе начиная с версии Windows XP.

WMIC позволяет выполнять следующие задачи:

– просматривать схемы WMI и запрашивать их классы и экземпляры (обычно с использованием <псевдонимов>, упрощающих работу с WMI);

– работать с локальным компьютером, удаленными компьютерами или выполнять команды сразу для нескольких компьютеров;

– настраивать псевдонимы и форматы вывода в соответствии с имеющимися потребностями;

– создавать и выполнять сценарии на основе WMIC.

Общий формат команды:

wmic *[глобальные параметры] < команда >*

Подробную информацию о допустимых параметрах и их командах, которые поддерживает WMIC, можно получить, используя следующие команды:

wmic /? - отобразить общую справку.
wmic /?:FULL - отобразить полную справку.

Рассмотри запрос получения информации о системной плате:

wmic BASEBOARD get /value | more,

где BASEBOARD – псевдоним класса системной платы ПЭВМ;

get – получение свойств псевдонима;

/value – параметр команды get, который возвращает значения в виде списка с названием свойства;

| more – отображения данных в постраничном режиме.

Приведем ещё несколько примеров запросов WMI:

wmic OS get OSArchitecture /value – отобразить архитектуру операционной системы (разрядность Windows);
wmic DISKDRIVE get Name, Size, Model – отобразить список физических дисков, содержащих название модели, имя в системе и размер;
wmic MEMORYCHIP get Devicelocator ,Capacity – отобразить сведения о размещении и емкости модулей памяти DIMM.

Основные псевдонимы, используемые для получения информации о конфигурации системы, представлены в табл. 1.2.

Таблица 1.2

Некоторые псевдонимы классов WMI

Псевдоним	Описание
BASEBOARD	Управление системной платой
BIOS	Управление BIOS
CDROM	Управление устройствами чтения компакт-дисков
CPU	Управление ЦП
DISKDRIVE	Управление физическими дисками
LOGICALDISK	Управление локальными запоминающими устройствами
MEMORYCHIP	Информация о микросхемах памяти
OS	Управление установленными операционными системами
SYSTEMSLOT	Управление точками физических соединений, включая порты, гнезда и периферийные устройства, а также специальными точками соединения

2.4. Системные ресурсы

Каждое периферийное устройство в зависимости от способа его подключения к ПК имеет право пользоваться системными ресурсами этого компьютера. Как правило, к системным ресурсам относятся:

- адреса портов ввода-вывода;
- адреса ячеек памяти;
- каналы запросов прерываний (IRQ);

Порты ввода-вывода позволяют установить связь между устройствами и программным обеспечением в компьютере. В современном компьютере 65 535 портов, пронумерованных от 0000h до FFFFh. Стандартное назначение адресов портов ввода-вывода Вы можете найти в рекомендованной литературе.

Каналы запросов прерывания (IRQ), или *аппаратные прерывания*, используются различными устройствами для сообщения системной плате (процессору) о необходимости обработки определенного запроса. В табл. 1.3 приведены назначения основных аппаратных прерываний в порядке убывания их приоритета.

3. Подготовка к работе

3.1. Изучить методические указания и рекомендованную литературу.

3.2. Подготовить ответы на контрольные вопросы.

Таблица 1.3

Аппаратные прерывания (в порядке убывания приоритета)

№ IRQ	Стандартная функция
0	Системный таймер
1	Контроллер клавиатуры
2	Резерв или второй контроллер прерываний
8	CMOS RTC – часы реального времени
9	Доступно (может использоваться вместо IRQ2)
10	Доступно
11	Доступно
12	PS/2 – мышь или доступно
13	Арифметический сопроцессор
14	Первичный IDE-канал
15	Вторичный IDE-канал или доступно
3	Последовательный порт (COM2, COM4)
4	Последовательный порт (COM1, COM3)
5	Звуковая плата или параллельный порт (LPT2)
6	Контроллер гибких дисков
7	Параллельный порт (LPT1)

4. Задание на выполнение работы

4.1. Создать и отладить программу на Ассемблере для получения информации о процессоре согласно варианту задания из табл. 1.4. Программа должна содержать проверку поддержки процессором команды CPUID и выводить информацию о количестве поддерживаемых функций CPUID и производителе процессора.

Вариант задания соответствует целой части суммы деления последней цифры зачетной книжки на два с единицей. Пример: последняя цифра зачетной книжки 7, тогда номер варианта находится из выражения N=7/2+1, в этом случае номер варианта равен 4.

Таблица 1.4

Варианты заданий

№ варианта	Выводимый параметр процессора
1	сигнатура процессора (в двоичном виде)
2	номер разработки (stepping)
3	модель
4	семейство
5	полное наименование

4.2. Используя утилиту *Сведения о системе (***MSinfo32.exe)** определить конфликтующие по используемым ресурсам устройства. В случае наличия таких устройств в отчет запишите информацию по одному из конфликтов для каждого ресурса: порт ввода-вывода, адрес памяти, прерывание IRQ.

4.3. Используя командную строку, ознакомится с возможностями приложения wmic.exe, вызвав справку и сформировав несколько запросов:

wmic /? – ознакомиться с командами WMIC.
wmic COMPUTERSYSTEM get /value – получить информацию о системе.
wmic BIOS get /value | more – получить информацию о BIOS.

4.4. Используя приложение wmic.exe сформировать запросы отображения серийных номеров накопителей информации, системной платы и оперативной памяти. Занести в отчет результаты сформированных запросов.

4.5. Используя системные утилиты MSinfo32.exe, Devmgmt.msc, Dxdiag.exe и средства WMI, заполнить соответствующие столбцы табл. 1.5 – 1.7.

Таблица 1.5

Сведения о системе

Характеристика	**Значение**
Изготовитель процессора	
Модель процессора	
Количество ядер процессора	
Частота ядра процессора	
Объем кэш памяти L1	
Объем кэш памяти L2	
Объем кэш памяти L3	
Сокет процессора	
Изготовитель системной платы	
Серия системной платы (Product)	
Модель системной платы	
Изготовитель видеоадаптера	
Модель видеоадаптера	
Объем видеопамяти	
Объем оперативной памяти	
Количество планок оперативной памяти	
Изготовитель оперативной памяти	
Поддерживаемая скорость передачи данных оперативной памятью	
Описание BIOS	
Версия операционной системы	
Версия DirectX	
Сетевое имя компьютера	

Таблица 1.6

Сведения о диске

Характеристика	**Значение**
Изготовитель	
Модель	
Объём	
Количество цилиндров	
Количество секторов	
Количество разделов на диске	

Таблица 1.7

Сведения о периферийных устройствах

Тип периферийного устройства	Название устройства	Выделяемые устройству ресурсы		
		Порты ввода/вывода	Прерывание	Ячейки памяти
Клавиатура				
Видеоадаптер				
Сетевой адаптер				
Последовательный порт				
Контроллер IDE ATA/ATAPI или SATA				
Хост-контроллер USB				

Примечание: в случае наличия в компьютере нескольких устройств одной категории в таблицу записывается информация обо всех устройствах.

5. Требования к отчёту

Отчёт должен содержать:

– титульный лист с указанием названия ВУЗа, кафедры, номера и темы лабораторной работы, а также Ф.И.О. студента, подготовившего отчёт;

– цель работы;

– листинг программы и результат её работы согласно заданию 4.1;

– информацию о наличие конфликтов в использованном ПЭВМ;

– информацию о серийных номерах согласно заданию 4.4.

– информацию об аппаратно-программной части использованного ПЭВМ, занесенную в таблицы 1.5 – 1.7.

6. Контрольные вопросы

6.1. Какими средствами можно определить аппаратную конфигурацию ПЭВМ?

6.2. Какие ресурсы могут быть выделены периферийным устройствам?

6.3. Каким устройствам выделяются стандартные ресурсы, а каким динамические?

6.4. В случае обнаружения конфликта по прерыванию при подключении нового устройства, какой номер прерывания ему лучше выдать? Как это сделать?

6.5. Какие прерывание и адреса портов ввода-вывода обычно выдаются первому последовательному порту?

6.6. Как получить список видеорежимов?

6.7. Как определить производителя видеоадаптера?

6.8. Как определить производителя и модель микропроцессора, установленного в ПЭВМ?
6.9. Для чего предназначена команда CPUID?
6.10. Как использовать команду CPUID? Какой алгоритм работы с командой CPUID?
6.11. Как определить поддержку процессором команд MMX и SIMD?
6.12. Для чего нужен WMI?
6.13. Какую информацию можно получить, используя программу WMIC?
6.14. Формат запроса через WMIC. Приведите примеры запроса к WMI через приложение WMIC.

7. Рекомендуемая литература

7.1. Мюллер, С. Модернизация и ремонт ПК, 19-е изд. [Текст]: Пер. с англ. / С. Мюллер. – М.: ООО «И.Д. Вильямс», 2011. – с.290…299.
7.2. Юров, В. И. Assembler: спец. справочник, 2-е изд. [Текст] / В.И. Юров. – СПб.: Питер, 2004. – с. 67…72.
7.3. Финогенов, К.Г. Использование языка Ассемблера: Учебное пособие для вузов [Текст] / К.Г. Финогенов. – М.: Горячая линия–Телеком, 2004. – с. 309…320.
7.4. CPUID – Идентификация CPU – Club155.ru [Электронный ресурс]. – Режим доступа: http://www.club155.ru/x86cmd/CPUID , свободный. – Загл. с экрана. – 17.02.2017.
7.5. CPUID Specification [Электронный документ] / Advanced Micro Devices – Режим доступа: https://support.amd.com/TechDocs/25481.pdf, свободный. – 20.02.2017.
7.6. Intel 64 and IA-32 Architectures Developer's Manual: Vol. 2A [Электронный документ] / Intel Corporation – Режим доступа: http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-2a-manual.html , свободный. – 20.02.2017.
7.7. About WMI (Windows) [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/en-us/library/aa384642(VS.85).aspx. свободный – 20.02.2017.
7.8. WMIC – использование инструментария управления Windows (WMI) в командной строке [Электронный документ]. – Режим доступа: http://ab57.ru/cmdlist/wmic.html. свободный – 20.02.2017.

Лабораторная работа №2
Конфигурационное пространство интерфейса PCI Express

1. Цель работы

Изучение структуры конфигурационного пространства и архитектуры современного компьютера с шиной PCI и PCI Express. Практическое овладение навыками определения конфигурации вычислительной машины.

2. Теоретический материал

2.1. Архитектура компьютера с шиной PCI Express

PCI Express (PCIe) – стандарт компьютерной шины, использующей программную модель шины PCI и высокоскоростную последовательную передачу данных. Стандарт является условно открытым, а его развитием занимается PCI Special Interest Group, в которую входят более 700 компаний по всему миру.

PCIe обеспечивает большую пропускную способность и более высокую масштабируемость по сравнению с шиной PCI. При этом PCIe имеет программную совместимость с интерфейсом PCI.

В основе интерфейса PCIe лежит двунаправленное последовательное соединение типа «точка-точка», которое называется линией. Физически соединение организована на базе двух дифференциальных пар, соединяющих приёмопередатчики двух устройств.

Пропускная способность шины PCIe зависит от количества используемых линий, частоты синхронизации и алгоритма кодирования передаваемых данных. Пропускная способность интерфейса PCIe может составлять от 0,5 ГБ/с до 60 ГБ/с и более.

Информация в PCIe передается в пакетах, которые могут содержать данные, адреса, служебные запросы и т.д., причем все виды пакетов передаются по тем же линиям, что и пакеты с данными в отличие от интерфейса PCI.

В интерфейсе PCIe реализована поддержка технологии качества обслуживания QoS: коммутатор PCIe может устанавливать приоритеты пакетов для критически важных по времени потоков данных (например, видеопоток в реальном времени).

Пример логической топологии интерфейса PCIe показана на рис. 2.1. Вершиной иерархией PCIe является процессор. Интерфейс между процессором и шинами PCIe может содержать несколько компонентов (процессорную шину, шину памяти и т.д.) и даже несколько микросхем. В совокупности эта группа называется **Корневым комплексом** (Root Complex). Он находится на вершине дерева PCI и действует от имени процессора для связи с остальной системой. Как пра-

вило, корневой комплекс содержит несколько внутренних мостов, чтобы увеличить количество портов. Физически корневой комплекс может быть реализован как отдельное устройство или интегрирован в процессор и чипсет системной платы. Допускается существование нескольких корневых комплексов в многопроцессорной системе. В этом случае корневые комплексы реализуются на чипсетах соответствующих процессоров, а связь между ними осуществляется через межпроцессорную шину, но адресация у них общая.

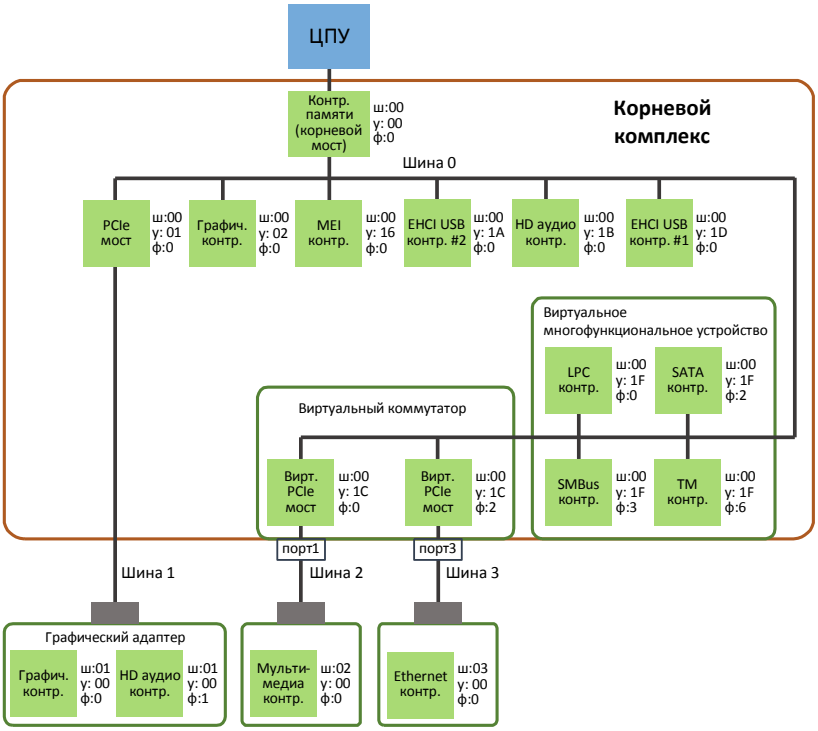

Рис. 2.1 – Топология системной платы ASRock H61M-DSG с процессором Intel Core i5-2400 и двумя внешними устройствами

Коммутатор (switch) обеспечивает разветвление выходов и возможность подключения большого количества устройств к одному порту PCIe. Он работает как маршрутизатор и на основе адреса распознает куда должен быть отправлен пакет. Внутри коммутатор реализуется как шина с несколькими виртуальными мостами.

Мост (bridge) обеспечивают связь с другими шинами, такими как PCI, PCI-X или даже другими шинами PCIe.

Оконечные устройства – это устройства в топологии PCIe, которые не являются коммутаторами или мостами, и выступают в качестве инициаторов и получателей транзакций на шине. Они располагаются в нижней части ветвей топологии дерева и реализуют только один восходящий порт (обращенный к корню). Согласно спецификации, существуют два вида оконечных устройств: *«устаревшее» оконечное устройство* (legacy endpoint), которое разработано на базе интерфейсов PCI или PCI-X, и *«родное» оконечное устройство PCIe* (native endpoint). Отличие этих устройств заключается в отсутствии поддержки новых возможностей интерфейса PCIe старыми устройствами. Оконечными устройствами для интерфейса PCIe являются сторонние контроллеры различных компонентов ЭВМ или интерфейсов, например, графический контроллер или контроллер USB интерфейса.

Согласно стандарту, в одной системе может существовать до 256 шин. Нулевой номер шины аппаратно назначается шине корневого комплекса. Эта виртуальная шина может содержать несколько встроенных оконечных устройств и виртуальных мостов, которым производитель аппаратно уже задал номера устройств и функций.

PCIe позволяет подключать до 32 устройств к одной шине. Каждое подключаемое устройство должно реализовать функцию с номером 0 и может содержать набор из восьми функций. Когда в одном устройстве реализованы две или более функции, такое устройство называется многофункциональным. При этом номера функций не обязательно должны назначаться последовательно.

В рассматриваемом примере топологии системной платы (рис. 2.1) большая часть устройств интегрирована в корневой комплекс, при этом в системе существуют два графических контроллера (один интегрирован в процессор, другой реализован в виде дискретного адаптера). У корневого комплекса задействованы два порта: один из которых для мультимедиа контроллера, подключаемого по средствам карты расширения, другой для Ethernet контроллера, установленного на системной плате.

2.2. Конфигурационное пространство

Для автоматической идентификации и конфигурирования каждая функция каждого устройства с интерфейсом PCI и PCIe содержит специальный набор регистров, который называют ***конфигурационным пространством*** (КП). КП PCI-устройств имеет размер 256 байт, PCI-X 2.0 и PCIe – 4096 байт. Для сохранения совместимости со старыми устройствами первые 256 байт КП PCIe соответствуют интерфейсу PCI.

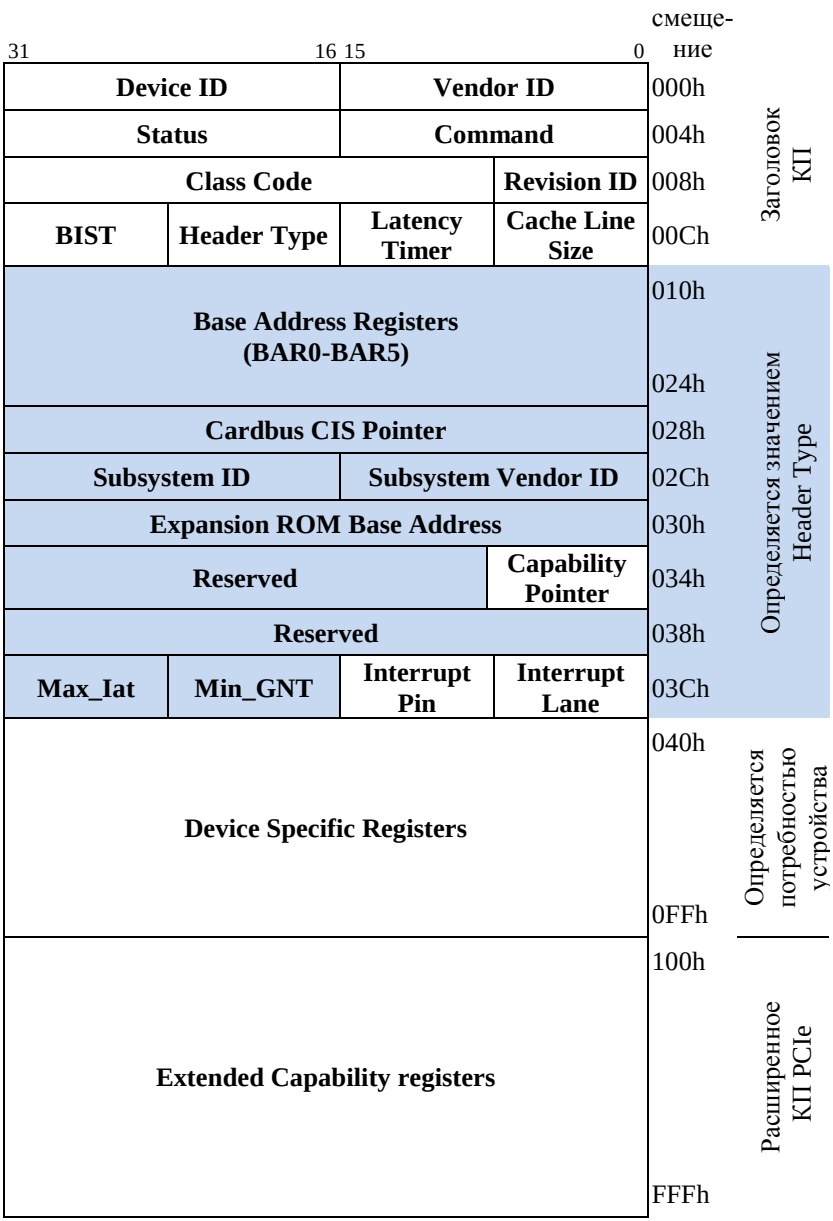

Рис 2.2 – Структура конфигурационного пространства функции устройства типа 0

Первые 64 байта КП каждой функции содержат структуру называемой ***заголовком***. Регистры заголовка КП имеет следующие назначение:

Vendor ID – идентификатор производителя устройства, назначаемый PCI SIG.

Device ID – идентификационный номер устройства, назначаемый производителем.

Command – регистр команд, служит для управления поведением устройства на шине PCI. Для PCIe устройств большинство бит данного регистра не используются и должны быть сброшены в «0».

Status – регистр состояния, служит для определения состояния и свойств устройства.

Revision ID – версия продукта, назначенная производителем.

Class Code – код класса, определяющий основную функцию устройства. Старший байт (адрес 0Bh) определяет базовый класс; средний – подкласс; младший – программный интерфейс.

Cache Line Size (RW) – регистр размера строки кэша для обмена данными с функцией устройства имеет одноименное назначение и устанавливается системным ПО низкого уровня и операционной системой. По этому параметру инициатор определяет, какой командой чтения воспользоваться (обычное чтение, чтение строки или множественное чтение). Ведомое устройство использует этот параметр для поддержки пересечения границ строк при пакетных обращениях к памяти. По сбросу регистр обнуляется. Этот регистр реализуется устройствами PCIe в целях обратной совместимости, но не влияет на какие-либо функциональные возможности устройств PCIe.

Latency Timer (RW) – регистр основного таймера задержки задает значение таймера, ограничивающего длину транзакции при снятии сигнала GNT#. Значение указывается в виде числа тактов шины, часть битов может не допускать изменения (обычно младшие три бита неизменны, так что таймер программируется с дискретностью в 8 тактов). Основной таймер задержки не применяется по отношению к CPIe. Данный регистр должен быть аппаратно сброшен в «0».

Header Type – тип заголовка (бит [0] определяет класс PCI функции, бит [7] является признаком многофункциональности устройства (если он установлен)).

BIST (RW) служит для управления встроенным самотестированием устройства (Built-in Self Test). Реализация регистра необязательна. Устройства, которые не поддерживают тест BIST, должны всегда возвращать значение 0.

Существуют два основных класса PCI функций, определяемые их типами в заголовке КП: тип 0 и тип 1. Тип определяется значением младшего бита поля *Header Type*. Тип 1 соответствует устройству, которое является мостом. Тип 0 соответствует всем остальным устройствам. Значение типа влияет на последующую структуру КП со смещением от 10h до 3Fh. Данная область КП для оконечного устройства содержит информацию о базовых адресах устройства в памяти или ввода-вывода, расширенных идентификаторах производителя и устройства, используемых линиях прерываний, диапазоне времени ожидания устройством. Для моста эта область содержит информацию о базовых адресах устройства в памяти или ввода-вывода, номерах родительской и дочерней шины, состоянии и характеристиках дочерней шины и самого моста.

Состав области специальных регистров устройства зависит от функциональности самого устройства и может содержать информацию о:

- механизме запросов прерываний (MSI) через сообщение;
- возможностях управления питанием устройства (PMI);
- базовых возможностях PCIe устройства;
- специфических данных от производителя.

Регистры каждой функциональности могут размещается в любом месте данной области, но базовое их смещение в КП должно находится или в регистре *Capability Pointer* (для первой функциональности) или в регистре *Next Item Pointer* (предыдущей структуры функциональности устройства). Структура каждой функциональности определяется по значению байта, куда указывает смещение. Этот регистр называется *Capability ID*. Ознакомиться со структурой каждой функциональности можно по рекомендуемой литературе [7.3].

Область регистров расширенных возможностей PCIe может содержать информацию о:

- серийных номерах устройства;
- ошибках;
- виртуальных каналах;
- необходимой мощности по питанию.

2.3. Доступ к конфигурационному пространству

Существует два способа доступа к конфигурационному пространству устройств с интерфейсом PCIe:

1) PCI-совместимый механизм конфигурирования, использующий порты ввода-вывода;

2) расширенный конфигурационный механизм, обращающийся к устройству через память.

Первый способ использовался ещё с интерфейсом PCI и позволяет обратиться только к первым 256 байта КП каждой функции. В данном механизме используются два 32-битных порта ввода-вывода, которые закреплены за регистрами корневого комплекса: порт конфигурации адреса 0CF8h (CONFIG_ADDRESS) и порт конфигурации данных 0CFCh (CONFIG_DATA). В порт конфигурации адреса необходимо поместить 4-х байтное число, соответствующее номерам шины, устройства, функции и смещения поля в КП, к которому хотим обратиться. Пример:

```
mov   dx,   0CF8h       ; Задание порта конфигурации адреса.
mov   eax, 80040000h    ; Формирование адреса области КП устройства:
                        ; шина 4, устройство 0, функция 0, смещение в КП 0.
out   dx,  eax          ; Занесение в порт интересующего адреса.
```

Порт конфигурации данных используется для записи и считывания данных размером до 4 байт по указанному адресу. Пример:

```
mov   dx, 0CFCh         ; Задание порта конфигурации данных.
in    ax, dx            ; Считывания 2-х байт из КП.
```

Получив адрес функции и команду через порты корневой комплекс транслирует их в конфигурационные транзакции для выбранного устройства. Подробное описание формата адреса и примеры работы с портами можно найти в рекомендуемой литературе [7.2].

Расширенный конфигурационный механизм использует отображение регистров конфигурационного пространства (4 КБ) каждой функции в плоском адресном пространстве памяти. Для этого в памяти выделяется 256 МБ. При такой реализации адрес ячейки памяти определяет конфигурационный регистр, в который осуществляется доступ, а данные в этой ячейке памяти – содержимое адресуемого регистра [7.1].

Адрес ячейки формируется из базового адреса, относительно которого в памяти размещаются КП всех PCIe устройств, адресов шины, устройства, функции и смещения поля в КП. Базовый адрес назначается системным ПО низкого уровня при каждом сбросе [7.5].

В связи с тем, что современные версии операционной системы Windows блокируют прямой доступ к портам ввода-вывода, а базовый адрес для расширенного механизма находится за пределами 1 Мбайта памяти, пользовательские приложения на Ассемблере не могут работать напрямую с КП PCIe. Для запуска таких приложений необходимо использовать «чистый» DOS. Доступ к любому устройству из ОС Windows возможен если обращаться к нему на уровне ядра через драйвер виртуального устройства.

2.4. Программы для просмотра КП PCI/PCIe

На данный момент существует множество утилит и пакетов программ, способных показывать информацию из конфигурационных пространств PCI/PCIe устройств. Далее приводятся некоторые из них.

Утилита **PCIUtils** выводит в консоль информацию обо всех PCI/PCIe устройствах: адреса, названия функций, идентификаторы производителя и продукта, дамп КП каждой функции, дерево шин.

Программа **RWEverything** (Read & Write utility) обеспечивает доступ ко многим аппаратным средствам компьютера, в том числе к PCI/PCIe устройствам. Данная программа позволяет не только показывать актуальную информацию об устройствах, но и редактировать её, что может привести к сбою системы при некорректном редактировании.

WinDriver кроссплатформенное средство разработки драйверов для устройств на интерфейсе USB, PCI и PCIe. WinDriver включает в себя графическую среду разработки, API-интерфейсы, средства отладки и диагностики. Пакет **WinDriver** имеет бесплатную полнофункциональную 30-дневную оценочную версию.

MindShare Arbor – это инструмент отладки, проверки, анализа и изучения компьютерных систем, который позволяет пользователю считывать и записывать данные в ячейки памяти, порты ввода-вывода или в конфигурационное адресное пространство PCI. Выводимые данные в программе представляются в упрощённом или графическом стилях. Программа Arbor имеет бесплатную полнофункциональную 14-дневную оценочную версию.

Бесплатная программа **TeleScan PE** [7.4] от Teledyne LeCroy для чтения, анализа, изучения и редактирования значений регистров КП PCIe устройств. Аппаратные продукты от компании Teledyne LeCroy одобрены PCI SIG в качестве стандартных инструментов для тестирования соответствия устройств спецификации PCIe 3.0.

При запуске **TeleScan PE** создает дерево устройств и считывает КП каждого обнаруженного PCI/PCI-X/PCIe устройства. В любое время после запуска пользователь может заставить **TeleScan PE** прочитать заново значения регистров КП. Также пользователь может менять содержимое регистров КП устройств на своё усмотрение. Изменения запишутся в устройство после нажатия кнопки «Redo».

3. Подготовка к работе

3.1. Изучить методические указания и рекомендованную литературу.

3.2. Подготовить ответы на контрольные вопросы.

4. Задание на выполнение работы

4.1. Ознакомится с программой TeleScan PE. Определить конфигурацию ПК с шиной PCIe с помощью данной программы. Составить топологию (схему) подключения к процессору компонентов ПК, использующих шину PCIe (пример такой схему показан на рис. 2.1).

4.2. Изучить поля заголовка конфигурационного пространства функции PCIe устройств. Записать значения полей заголовка КП одной функции устройства, согласно варианту задания. Вариант задания определяется по последней цифре зачетной книжки и берётся из таблицы 2.1.

Таблица 2.1

Варианты заданий

№ варианта	Устройство	№ варианта	№ атрибута
0	Корневой мост	5	SATA (IDE) контроллер
1	Графический контроллер	6	MEI-контроллер
2	Аудио контроллер	7	PCIe мост (00:01.0)
3	Сетевой контроллер	8	Любой порт корневого комплекса
4	USB контроллер (любой)	9	LPC контроллер

5. Требования к отчёту

Отчет должен содержать:

– титульный лист с указанием названия ВУЗа, кафедры, номера и темы лабораторной работы, а также фамилии и инициалов студента, подготовившего отчёт;

– цель работы;

– топологию исследуемого ПК с шиной PCIe;

– вариант задания;

– наименования и значения полей заголовка КП функции устройства согласно варианту задания или снимок экрана монитора с окном КП;

– выводы по работе.

6. Контрольные вопросы

6.1. Для чего предназначена шина PCIe? Какова ее роль в архитектуре современного ПК?

6.2. Каковы основные характеристики шины PCIe?
6.3. Сколько устройств можно подключать к шине PCIe?
6.4. Как обратиться к конфигурационному пространству PCIe устройства?
6.5. Какие поля имеются в конфигурационном пространстве устройств PCIe?
6.6. Какой формат регистра Class Code конфигурационного пространства PCIe? Какие возможны классы устройств?
6.7. Поясните назначение регистра Header Type?
6.8. Как определить серийный номер оконечного устройства PCIe?
6.9. Для чего предназначена программа TeleScan PE? Какие операции можно выполнить с её помощью?
6.10. Какой размер имеет конфигурационное пространство PCIe устройств?

7. Рекомендуемая литература

7.1. Петров, С.В. Шины PCI, PCI Express. Архитектура, дизайн, принципы функционирования [Текст] / С.В. Петров. – СПб.: БХВ-Петербург, 2006. – с. 319-343.
7.2. Несвижский, В. Программирование аппаратных средств в Windows [Текст]: 2-е изд., перераб. и доп. / В. Несвижский. – СПб.: БХВ-Петербург, 2008. – с. 339-366.
7.3. Jackson, Mike. PCI Express Technology / M. Jackson, R. Budruk. / MindShare, Inc., 2012. – p. 39-54, 85-117.
7.4. Teledyne LeCroy – Protocol Analyzer – TeleScan PE Software [Электронный ресурс] / Teledyne LeCroy. – Режим доступа: http://teledynelecroy.com/protocolanalyzer/protocoloverview.aspx?seriesid=447& capid=103&mid=511 , свободный – 17.07.2017.
7.5. Memory-mapped Configuration [Электронный документ] / IC Book – Режим доступа: http://icbook.com.ua/press/memory_mapped_configuration, свободный – 17.07.2017.

Лабораторная работа №3
Исследование S.M.A.R.T.-атрибутов жестких дисков

1. Цель работы

Практическое овладение навыками получения и анализа S.M.A.R.T.- атрибутов жестких дисков.

2. Теоретический материал

2.1. Технология S.M.A.R.T.

S.M.A.R.T. (Self-Monitoring Analysis and Reporting Technology) – это технология внутренней оценки состояния диска, и механизм предсказания возможного выхода из строя жесткого диска. S.M.A.R.T. производит наблюдение за основными параметрами накопителя. Эти параметры называются атрибутами.

Атрибуты S.M.A.R.T. – особые характеристики, которые используются при анализе состояния и запаса производительности накопителя. Каждый производитель имеет свой характерный набор атрибутов и может свободно вносить изменения в этот набор в соответствии со своими требованиями.

Одной из основных функций системы мониторинга является возможность самоконтроля состояния жесткого диска. Выполнение данной тестовой процедуры может быть осуществлено как самим накопителем, не занятым клиентским заданием, так и пользователем, осуществляющим проверку атрибутов S.M.A.R.T. посредством специализированного программного обеспечения. В любом случае, чтобы начать принудительный процесс проверки, следует подать интерфейсную команду Smart Execute Offline Immediate. По прошествии некоторого времени, требуемого для получения финального результата, накопитель сохраняет полученные данные в специализированных атрибутах и журналах. Результаты тестирования используются накопителем для сравнения с полученными ранее данными. Таким образом, можно наблюдать тенденцию изменения атрибутов, что позволит делать выводы о примерном выходе из строя жесткого диска в целом.

Каждый атрибут имеет свой уникальный идентификатор – **ID**. Он характеризует некоторую реальную величину, например, количество изношенных секторов или общее время работы, на основании которой можно делать выводы о надежности конструкции в целом. Большинство жестких дисков, поддерживающих S.M.A.R.T., обычно имеют от 3 до 30 атрибутов.

Значения всех атрибутов надежности (value) обычно находятся в диапазоне от 1 до 253 включительно, но могут быть и в другом диапазоне. При производстве жесткого диска каждый атрибут получает

максимальное значение. Постепенно, по мере износа накопителя, значения атрибутов надежности уменьшаются. Соответственно, высокое значение атрибутов говорит о низкой вероятности выхода жесткого диска из строя, и, наоборот, низкое значение атрибутов – о низкой его надежности и высокой вероятности скорого отказа.

Диапазон изменения атрибутов не стандартизирован. Каждый производитель вносит свою лепту в данную технологию. Так, например, для продуктов, произведенных компанией Hitachi Data Storage, максимальная величина каждого атрибута составляет 100 единиц. Для Samsung это число равно 253. Наибольшую путаницу внесли инженеры компании Western Digital, поскольку для своих продуктов они используют довольно странную методику измерений. Так, верхняя граница первого атрибута надежности составляет 200 единиц, а остальных – 100.

При работе накопителя ведутся журналы ошибок и значений атрибутов, в которых хранится информация о нескольких последних ошибках и **худших значениях** атрибутов с момента первого запуска жесткого диска.

Для каждого атрибута надежности разработчиками жестких дисков определяется **пороговое значение**, называемое Threshold, по достижении которого устройство можно считать небезопасным для хранения данных.

Помимо текущего значения, описывающего состояния атрибутов, имеются **необработанные (raw) значения**, которые несут определенный смысл для каждого атрибута. Например, необработанное значение атрибута *Power-On Hours* (Наработка в часах) является счетчиком единиц времени (часов, минут, секунд и т.п.), в течении которого жесткий диск находился в работающем состоянии.

Также каждый атрибут имеет **флаги**, указывающие на назначение атрибута; атрибут может иметь несколько флагов одновременно. Флаги атрибута:

- LC – атрибут, непосредственно описывающий надежность диска.
- PR – атрибут, отражающий производительность диска.
- ER – атрибут, отражающий частоту появления ошибок.
- EC – означает, что атрибут используется как счетчик каких-либо событий.
- SP – атрибут, значения которого автоматически сохраняются и восстанавливаются каждый раз, когда производятся тесты S.M.A.R.T.
- OC – значения этого атрибута вычисляются во время проведения тестов реального времени.

Краткое описание основных атрибутов надежности S.M.A.R.T. приведено в табл. 3.1. Данная таблица включает не полный список всех атрибутов S.M.A.R.T., более полный список можно найти в рекомендуемой литературе.

Таблица 3.1

Основные S.M.A.R.T. атрибуты накопителей

№ атрибута	Имя атрибута	Описание
1	Raw Read Error Rate	уровень ошибок при чтении данных с диска, происхождение которых обусловлено аппаратной частью диска.
3	Spin Up Time	время раскрутки шпинделя диска из состояния покоя до рабочей скорости.
4	Start/Stop Count	полное число запусков/остановов шпинделя. У дисков некоторых производителей (Seagate, например) – счетчик включения режима энергосбережения. В поле raw value хранится общее количество запусков/остановок диска.
5	Reallocated Sectors Count	количество переназначенных секторов. Когда диск обнаруживает ошибку чтения/записи, он помечает сектор «переназначенным», и переносит данные в специально отведенную область. Чем меньше значение, тем хуже состояние поверхности дисков. Поле raw value содержит общее количество переназначенных секторов.
7	Seek Error Rate	частота ошибок при позиционировании блока головок. Чем их больше, тем хуже состояние механики и/или поверхности жесткого диска.
9	Power-On Hours	число часов, проведённых во включённом состоянии. В качестве порогового значения для него выбирается паспортное время наработки на отказ.
10	Spin-Up Retry Count	число повторных попыток раскрутки дисков до рабочей скорости, в случае если первая попытка была неудачной. Ненулевое значение свидетельствует о проблемах в механической части накопителя.
12	Device Power Cycle Count	количество полных циклов включения-выключения диска.
194 (231)	Temperature	показания встроенного термодатчика.

Продолжение таблицы 3.1

№ атрибута	Имя атрибута	Описание
197	Current Pending Sector Count	число подозрительных или нестабильных секторов, являющихся кандидатами на замену. Они не были ещё определены как плохие, но считывание их отличается от чтения стабильного сектора. В случае успешного последующего прочтения сектора он исключается из числа кандидатов. В случае повторных ошибочных чтений накопитель пытается восстановить его и выполняет операцию переназначения.
198	Uncorrectable Sector Count	число неисправимых ошибок при обращении к сектору. В случае увеличения числа ошибок велика вероятность критических дефектов поверхности и/или механики накопителя.
199	UltraDMA CRC Error Count	число ошибок, возникающих при передаче данных по внешнему интерфейсу.
200	Write Error Rate / Multi-Zone Error Rate	показывает общее количество ошибок, происходящих при записи сектора. Может служить показателем качества поверхности и механики накопителя.

2.2. Получение информации о S.M.A.R.T. атрибутах

Для получения значений S.M.A.R.T. атрибутов жёстких дисков на сегодняшний день существует множество платных и бесплатных программ, отличающихся дополнительными функциями при обработке данных. К наиболее известным программам данного типа относятся: **HDDLife**, **Hard Drive Inspector**, **HD Tune**, **PCS**, **ActivSMART** и др. Однако с помощью несложных команд возможно написание собственных программ для получения значений атрибутов и предсказания выхода из строя жёстких дисков. Для получения значения атрибутов необходимо обратиться или непосредственно к жёсткому диску или к драйверу мини-порта контроллера SCSI.

Пример алгоритма программы для получения значений атрибута показан на рис. 3.1 и 3.2. Данная программа опрашивает первые десять накопителей и выводит значения атрибута по каждому накопителю в текущее консольное окно. В случае отсутствия накопителей или запуска программы без прав администратора выводится сообщение об ошибке. В разделе 4 приведён пример листинга 32-х разрядной программы на языке Ассемблера, которая получает значения атрибута

температуры и выводит их в консольное окно. В связи с различной интерпретацией текущего значения температуры у разных производителей ориентироваться необходимо на необработанное значение атрибута.

Чтение S.M.A.R.T. атрибутов можно выполнить с помощью функции **DeviceIoControl**, которая отправляет управляющий код непосредственно указанному драйверу устройства, заставляя соответствующее устройство выполнять указанную операцию.

Перед выполнением функции **DeviceIoControl** необходимо прочитать дескриптор (манипулятор или handle) диска «как файл» функцией **CreateFile**. Данная функция имеет следующие параметры:

1. *lpFileName* – указатель на символьную строку, устанавливающую имя открываемого объекта. Строка должна заканчиваться нулем;
2. *dwDesiredAccess* – режим доступа к объекту;
3. *dwShareMode* – режим совместного использования;
4. *lpSecurityAttributes* – параметры безопасности, который устанавливает возможность наследования возвращённого дескриптора дочерними процессами. Чаще всего данный параметр равен нулю;
5. *dwCreationDisposition* – выполняемое действие;
6. *dwFlagsAndAttribute* – атрибуты и флаги файла;
7. *hTemplateFile* – дескриптор шаблона файла.

В случае успешного выполнения функция **CreateFile** вернёт в регистр EAX дескриптор (хендл) объекта, имя которого было указано в первом параметре, в случае неудачи вернёт значение INVALID_HANDLE_VALUE (а именно -1 или 0FFFFFFFFh).

Более подробную информацию о параметрах функции можно найти в рекомендованной литературе, а пример получения дескриптора диска с помощью функции **CreateFile** приведен в процедуре **InitSMART** программы **SMART**.

Данная функция работает, если программа запущена с правами администратора, в противном случае функция CreateFile завершается с ошибкой и возвращает значение – INVALID_HANDLE_VALUE.

После успешного открытия устройства можно работать с ним, используя функцию **DeviceIoControl**. Она принимает восемь параметров:

1. *hDevice* – имя дескриптора устройства, над которым должна выполниться операция;
2. dwIoControlCode – управляющий код для операции;
3. lpInBuffer – указатель на буфер ввода данных, который содержит данные необходимые, чтобы выполнить операцию;
4. nInBufferSize – размер буфера ввода данных, в байтах;

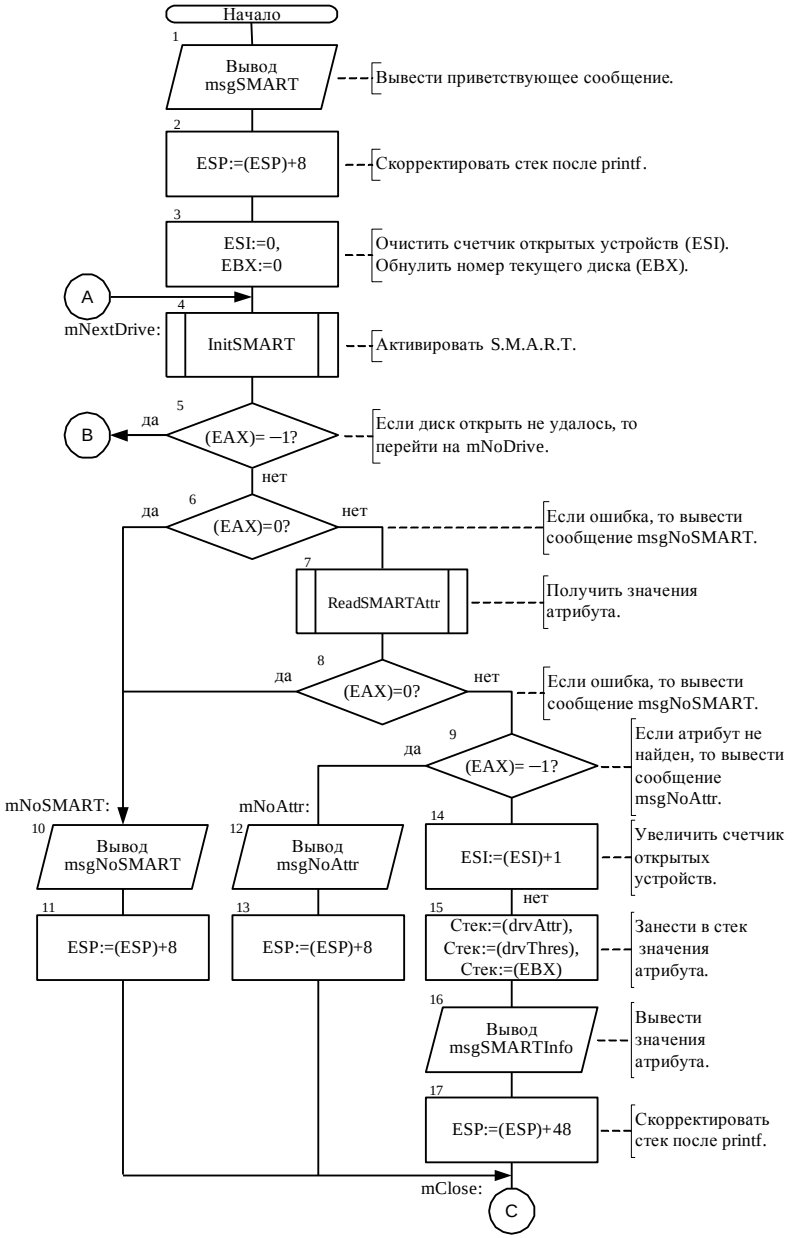

Рис. 3.1 – Алгоритм получения значений S.M.A.R.T. атрибута накопителей

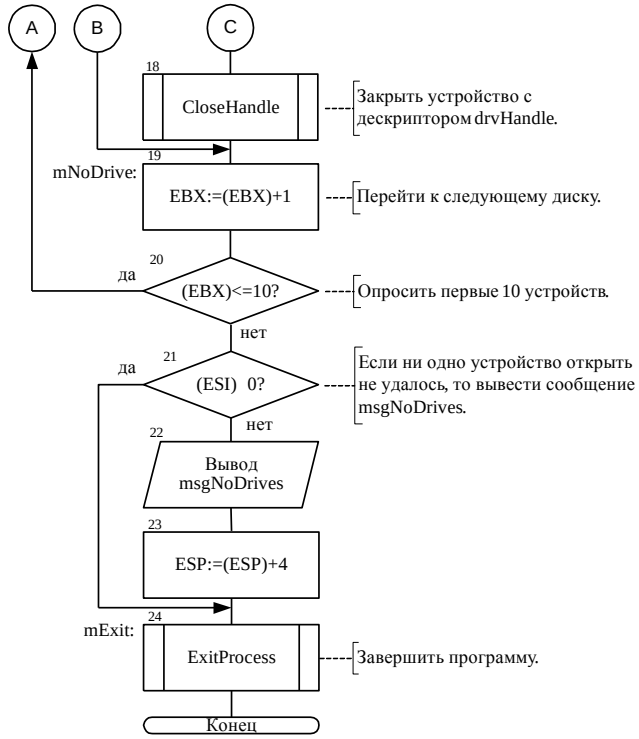

Рис. 3.2 – Продолжение алгоритма получения значений S.M.A.R.T. атрибута накопителей

5. lpOutBuffer – указатель на буфер вывода данных, который должен получить данные, возвращенные операцией;

6. nOutBufferSize – размер буфера вывода данных, в байтах;

7. lpBytesReturned – указатель на переменную, которая получает размер данных, сохраненных в буфере вывода данных, в байтах;

8. lpOverlapped – указатель на структуру OVERLAPPED необходимую для асинхронных операций.

Для передачи жесткому диску команды активации S.M.A.R.T. в качестве управляющего кода необходимо указать константу DFP_SEND_DRIVE_COMMAND (0007C084h), а для получения данных от диска константу DFP_RECEIVE_DRIVE_DATA (0007C088h). Сами команды S.M.A.R.T. указываются в буфере ввода данных, который удобно оформить в виде структуры данных (табл. 3.2). Для активации S.M.A.R.T. в поле bFeaturesReg необходимо занести константу IDE_SMART_ENABLE (0D8h), а для получения текущего и порогово-

го значений атрибута константы IDE_SMART_READ_ATTRIBUTES (0D0h) и IDE_SMART_READ_THRESHOLDS (0D1h) соответственно. Номер диска/головки IDE для поля bDriveHeadReg вычисляется по формуле [0A0h or ((Drive and 1) shl 4)].

Таблица 3.2

Структура буфера ввода данных для команды DeviceIOControl

Имя структуры записи	Элемент структуры	Размер элемента, байт	Значение элемента	Описание
SendCmd-InParams (SCIP)	cBufferSize	4	512	размер буфера в байтах
	irDriveRegs		IDERegs	структура регистров диска
	bDriveNumber	1		физический номер диска
	bReserved	3		зарезервировано
	dwReserved	16		зарезервировано
	bBuffer	1		входной буфер
IDERegs	bFeaturesReg	1		регистр подкоманды S.M.A.R.T.
	bSectorCountReg	1	1	регистр количества секторов диска
	bSectorNumberReg	1	1	регистр номера сектора диска
	bCylLowReg	1	4Fh	младший разряд номера цилиндра диска
	bCylHighReg	1	C2h	старший разряд номера цилиндра диска
	bDriveHeadReg	1		регистр диска/головки
	bCommandReg	1	B0h	регистр фактической команды
	bReserved	1	0	зарезервировано

Структура буфера вывода данных представлена в табл. 3.3. Так как буферы вывода данных для текущих и пороговых значений имеют одинаковую структуру, то желательно создать два буфера с разными именами. Значения всех атрибутов возвращаются в массиве bBuffer со смещение в 2 байта. Его состав для текущих и пороговых значений показан в табл. 3.4.

Таблица 3.3

Структура буфера вывода данных для команды DeviceIOControl

Имя структуры записи	Элемент структуры	Размер элемента, байт	Описание
SendCmd-OutParams (SCOP)	cBufferSize	4	размер буфера в байтах
	DriverStatus	DriverStatus	структура состояния диска
	bBuffer	массив	буфер произвольной длины для сохранения данных, прочитанных с диска
DriverStatus	bDriverError	1	код ошибки драйвера
	bIDEStatus	1	содержание регистра ошибки
	bReserved	1	зарезервировано
	dwReserved	2	зарезервировано

Таблица 3.4

Структура возвращаемых данных в bBuffer

Имя структуры записи	Элемент структуры	Размер элемента, байт	Описание
Drive-Attribute	bAttrID	1	идентификатор атрибута
	wStatusFlags	2	флаги состояния
	bAttrValue	1	текущее нормализованное значение
	bWorstValue	1	худшее значение
	bRawValue	6	текущее ненормализованное значение
	bReserved	1	зарезервировано
Attr-Threshold	bAttrID	1	идентификатор атрибута
	bWarranty-Threshold	1	пороговое значение
	bReserved	10	зарезервировано

При выводе информации в консольное окно удобно использовать API-функцию **printf**. Эта функция выводит отформатированный текст согласно указанному формату в стандартный выходной поток. Функция может иметь несколько параметров: первый параметр – *указатель на буфер* (строку), который будет форматироваться; следующие параметры – *аргументы, которые будут подставляться в исходную строку*. При выводе **printf** начинает печатать символы, которые находит по адресу, указанному в первом параметре, символ за симво-

лом. Когда в этом процессе попадается символ процента (%), то **printf** знает, что это спецификатор формата – специальный набор символов, который задает, как надо вывести число. Следующий по порядку параметр, указанный в **printf**, выводится так, как указано в спецификаторе формата. Затем процесс обработки символов (вывод их на печать) первого параметра продолжается. Если опять встретится знак процента, то будет выведен следующий параметр в указанном формате, после чего вывод строки продолжится до тех пор, пока не встретиться нулевой символ (символ с кодом 0).

Формат задается с помощью *спецификатора*, начинающегося со знака процента. Существует достаточно много опций для задания формата спецификатора из которых рассмотрим только три: модификатор размера, модификатор типа и опция заполнения лидирующими нулями. Размер зависит от типа модификатора, поэтому указывается совместно с ним. Размер определяет количество символов, которое будет выведено, однако если цифр в числе меньше, чем задано в спецификаторе, то число выведется с пробелами слева. Чтобы придать наглядный вид числу при выводе, можно воспользоваться опцией заполнения лидирующими нулями, которая задается указанием нуля после знака процента.

Тип спецификатора задается в виде одного символа и является обязательным в спецификаторе. В нашем случае используются несколько типов:

%u – выводит на печать целое десятичное число без знака;

%x или **%X** – выводит на печать целое шестнадцатеричное число строчными или прописными символами, соответственно.

Пример использования функции **printf** на Ассемблере:

```
.data
        msg     db 'Вы учитесь на %u курсе', 13, 10, 0
        course dw 4
.code
start:  movzx EBX, course
        call    printf, offset msg, EBX
        add     ESP, 4*2
```

Поскольку в Ассемблере параметры функции передаются через стек, то в указанном примере выполняется предварительное преобразование размера переменной *course* из байта в двойное слово, используя регистр EBX.

Если функция выполнена успешно, то в регистр EAX будет возвращена длина скопированной строки.

В отличие от других WinAPI-функций **printf** не знает сколько параметров может быть в неё отправлено, поэтому после её выполнения программист обязан освободит стек. Стек освобождается командой

ADD ESP, N,

где N – это количество освобождаемых байтов. В выше указанном примере через стек передаются два параметра, каждый из которых размером 4 байта.

2.3. Использование структур данных в Ассемблере

Структура – конструкция, объединяющая набор переменных разных типов. Переменные, образующие структуру, называются элементами структуры или полями.

Для использования структур в программе необходимо выполнить три действия:

1) описать структуру;
2) определить экземпляр структуры;
3) организовать обращение к элементам структуры.

Описать структуру в программе означает указать ее шаблон. Этот шаблон можно рассматривать лишь как информацию для транслятора о расположении полей и их значении по умолчанию. Описание структуры можно располагать в любом месте программы, но до описания конкретных структурных переменных. Транслятор, встретившись с описанием структуры, не транслирует её текст, т.е. не выделяет место в памяти, а запоминает приведенное описание, чтобы воспользоваться им в дальнейшем, если в программе встретится объявления переменных типа этой структуры.

Для описания структур данных в Ассемблере имеется директива STRUC. Общий форма описания структур данных выглядит следующим образом:

<имя структуры> STRUC
<описание полей структуры>
<имя структуры> ENDS

Здесь *<описание полей>* представляет собой последовательность директив описания данных DB, DW, DD и др. Их операнды определяют размер полей и, при необходимости, начальные значения. Этими значениями будут, возможно, инициализироваться соответствующие поля при определении структуры. Пример описания структуры:

```
point STRUC
        x       DW    0
```

```
        y       DW    0
        z       DW    0
        color   DB    3 DUP (?)
point ENDS
```

Для использования описанной с помощью шаблона структуры в программе необходимо *определить переменную* с типом данной структуры. Для этого используется следующая синтаксическая конструкция:

[имя переменной] имя структуры <[список значений]>,

где *имя переменной* – идентификатор переменной данного структурного типа;

список значений – заключенный в угловые скобки список начальных значений элементов структуры, разделенных запятыми. Если при определении новой переменной с типом данной структуры нет необходимости менять значения по умолчанию, которые записаны в шаблоне, то необходимо оставить пустым содержимое угловых скобок.

Используя выше указанный шаблон структуры можно инициализировать несколько переменных:

```
point1 point <>
point2 point <100, 200, 1, 255, 255, 255>
```

При этом значения полей переменной *point1* будут взяты из шаблона.

Для того чтобы сослаться в команде на поле некоторой структуры, используется специальный оператор – символ « . » (точка). Он используется в следующей синтаксической конструкции:

имя переменной . имя поля структуры,

где *имя переменной* – идентификатор переменной некоторого структурного типа;

имя поля структуры – имя поля из шаблона структуры.

Пример:

```
mov AX, point2.y
```

3. Подготовка к работе

3.1. Изучить методические указания и рекомендованную литературу.

3.2. Подготовить ответы на контрольные вопросы.

4. Задание на выполнение работы

4.1. Создать и отладить исполняемый модуль программы **SMART**, исходный код которой приведён далее. Добавить в исходный модуль программы недостающие комментарии.

4.2. Отредактировать исходный модуль программы **SMART** таким образом, чтобы она выводила значения атрибута в соответствии с вариантом задания. Варианты задания указаны в табл. 3.5.

4.3. По полученным значениям сделать выводы о производительности и «здоровье» жёсткого диска.

Таблица 3.5

Варианты заданий

№ варианта	№ атрибута	№ варианта	№ атрибута
1	1	7	10
2	3	8	12
3	4	9	197
4	5	10	198
5	7	11	199
6	9	12	200

```
TITLE SMART
;32-х разрядное приложение получения значений S.M.A.R.T. атрибута
;жесткого диска.
;Программа выводит в текущую консоль для каждого диска номер
;атрибута, егоописание и значения: текущее нормированное, худшее,
;пороговое и необработанное.

.386                                ;Расширенная система команд.
; Плоская модель памяти и стандартная модель вызова подпрограмм.
.MODEL FLAT, STDCALL
; Директивы компоновщику для подключения библиотек.
INCLUDELIB import32.lib             ;Работа с библиотекой ОС Kernel32.

;--- Внешние WinAPI-функции ---------------------------------------------
EXTRN CreateFileA:      PROC  ;Получить дескриптор устройства.
EXTRN DeviceIoControl:  PROC  ;Получить информацию об устройстве.
EXTRN CloseHandle:      PROC  ;Закрыть дескриптор устройства.
EXTRN printf:           PROC  ;Вывести текст по шаблону.
EXTRN RtlZeroMemory:    PROC  ;Очистить память.
EXTRN ExitProcess:      PROC  ;Завершить процесс.
```

```
;--- Константы ----------------------------------------------------------------
; Стандартные значения WinApi
GENERIC_READ    = 80000000h     ;допускается чтение
GENERIC_WRITE   = 40000000h     ;допускается запись
FILE_SHARE_READ   = 1           ;допускается чтение другими процессами
FILE_SHARE_WRITE  = 2           ;допускается запись другими процессами
OPEN_EXISTING     = 3           ;предписывает открывать устройство,
                                ;если оно существует

READ_ATTRIBUTE_BUFFER_SIZE = 512    ;размер буфера атрибутов
READ_THRESHOLD_BUFFER_SIZE = 512    ;размер буфера порог. значений
SMART_CYL_LOW = 4Fh             ;младший байт цилиндра для S.M.A.R.T.
SMART_CYL_HI    = 0C2h          ;старший байт цилиндра для S.M.A.R.T.

; Управляющие коды для функции DeviceIOControl
DFP_SEND_DRIVE_COMMAND = 0007C084h
DFP_RECEIVE_DRIVE_DATA = 0007C088h

; Коды запросов
IDE_SMART_ENABLE      = 0D8h    ;на активацию S.M.A.R.T.
IDE_SMART_READ_ATTRIBUTES = 0D0h ;на чтение атрибутов
IDE_SMART_READ_THRESHOLDS = 0D1h ;на чтение порогового значения
IDE_COMMAND_SMART         = 0B0h ;для работы со S.M.A.R.T.

ATTR_NAME  equ  <'Temperatura'>      ;название атрибута

;-- Структуры ---------------------------------------------------------------
; Структура записи регистров IDE
IDERegs STRUC
        bFeaturesReg     db ?    ;Регистр подкоманды SMART
        bSectorCountReg  db ?    ;Регистр количества секторов IDE
        bSectorNumberReg db ?    ;Регистр номера сектора IDE
        bCylLowReg       db ?    ;Младший разряд номера цилиндра IDE
        bCylHighReg      db ?    ;Старший разряд номера цилиндра IDE
        bDriveHeadReg    db ?    ;Регистр номера диска/головки IDE
        bCommandReg      db ?    ;Фактическая команда IDE
                         db ?    ;Зарезервировано (должен быть 0)
IDERegs ENDS

; Структура записи входных параметров для функции DeviceIOControl
```

```
SendCmdInParams STRUC
        cBufferSize     dd ?                    ;Размер буфера в байтах
        irDriveRegs     IDERegs <>
        bDriveNumber db ?                       ;Номер физ. диска
                        db 3 dup (?)            ;Зарезервировано
                        dw 8 dup (?)            ;Зарезервировано
        bInBuffer       label  byte             ;Входной буфер
SendCmdInParams ENDS

; Структура состояния диска
DriverStatus STRUC
        bDriverError db ?                       ;Код ошибки драйвера
        bIDEStatus   db ?                       ;Значение регистра ошибки IDE
                                                ;Контроллера при bDriverError = 1
                     db 2 dup (?)               ;Зарезервировано
                     dd 2 dup (?)               ;Зарезервировано
DriverStatus ENDS

; Структура записи параметров, возвращаемых функцией DeviceIOControl
SendCmdOutParams STRUC
        cBufferSize    dd ?          ;Размер буфера в байтах
        DrvStatus  DriverStatus <>   ;Структура состояния диска
        bOutBuffer  label byte       ;Буфер произвольной длины для хранения
                                     ;данных, полученных от диска
SendCmdOutParams ENDS

; Структура записи значений атрибутов
DriveAttribute STRUC
        bAttrID         db ?         ;Идентификатор атрибута
        wStatusFlags dw ?            ;Флаг состояния
        bAttrValue      db ?         ;Текущее нормализованное значение
        bWorstValue     db ?         ;Худшее значение
        bRawValue       db 6 dup (?) ;Текущее ненормализованное значение
                        db ?         ;Зарезервировано
DriveAttribute ENDS

; Структура записи порогового значения атрибута
AttrThreshold STRUC
        bAttrID                 db ?          ; Идентификатор атрибута
        bWarrantyThreshold db ?               ; Пороговое значение
                                db 10 dup (?) ; Зарезервировано
```

```
AttrThreshold ENDS

;--- Данные ------------------------------------------------------------------------
.DATA
  sDrive              db '\\.\PhysicalDrive'  ; имя устройства,
  cDrive              db '0', 0               ; включая его номер
  msgSMART            db 'Чтение значений S.M.A.R.T. атрибута '
                      db 'всех подключенных дисков...', 13, 10, 0
  msgNoSMART          db 13, 10, 'HDD%u не поддерживает '
                      db 'технологию S.M.A.R.T.!', 13, 10, 0
  msgNoAttr           db 13, 10, 'Атрибут ',ATTR_NAME,' для HDD%u '
                      db 'не найден!', 13, 10, 0
  msgNoDrives         db 13, 10, 'Не удалось открыть ни одного устройства '
                      db '(видимо, у Вас нет прав администратора)',13,10, 0
  msgSMARTInfo        db 13, 10, 'Значения атрибута для HDD%u',13,10
        db      'Attribute number = %u (', ATTR_NAME, ')', 13, 10
        db      ' Attribute value = %u', 13, 10
        db      '     Worst value = %u', 13, 10
        db      ' Threshold value = %u', 13, 10
        db      '    RAW value = %02X%02X%02X%02X%02X%02X (hex)',13,10,0

  hDrive        dd ?                    ;дескриптор устройства
  bReturned     dd ?                    ;число байт, возвращённых функцией

; Указатели на структуры
  drvAttr   DriveAttribute  <>      ; значений атрибута
  drvThres AttrThreshold    <>      ; пороговых значений атрибута
  SCIP    SendCmdInParams <>        ; входных параметров
  SCOP    SendCmdOutParams <>       ; выходных параметров
                  db 512 dup (?)    ; зарезервированное место для буфера

.CODE
;=== Активация S.M.A.R.T. для диска номер Drive ===============
; Возвращает EAX = 1 - успешное завершение, 0 - ошибка (нет прав),
;  -1 - диск не найден
InitSMART PROC
        ARG   Drive: DWORD

        mov     al, byte ptr Drive
        add     al, '0'         ;Преобразование номер диска в символ '0', '1' и т.д.
        mov     cDrive, al      ;Сохранение его в cDrive
```

```
                                ;(корректировка строки sDrive)

; Получение дескриптора диска
        call    CreateFileA, offset sDrive, GENERIC_READ or
GENERIC_WRITE, FILE_SHARE_READ or FILE_SHARE_WRITE, 0,
OPEN_EXISTING, 0, 0
        cmp     eax, -1             ;В случае ошибки выход из
        je      mISExit             ;процедуры с возвращением -1
        mov     hDrive, eax         ;Сохранение дескриптор в hDrive

; Подготовка буфера SCIP для активации S.M.A.R.T
        call    InitSMARTInBuf, Drive, IDE_SMART_ENABLE, offset SCIP, 0

; Активация S.M.A.R.T. для текущего диска
        call    DeviceIoControl, hDrive, DFP_SEND_DRIVE_COMMAND,
offset SCIP, size SCIP, offset SCOP, size SCOP, offset bReturned, 0
        test    eax, eax            ;Проверка успешного выполнение команды
        jz      mISExit             ;В случае ошибки возвращается 0,
        mov     eax, 1              ;иначе 1.
mISExit:    ret
InitSMART   ENDP

;=== Чтение значений S.M.A.R.T. атрибута  ====================
; DriveAttr и AttrThres – буферы для значений атрибута,
; Drive – номер диска.
; Возвращает EAX = 1 - успешное завершение, 0 - ошибка (нет прав),
; -1 - атрибут не найден.
ReadSMARTAttr PROC
        ARG     Drive: DWORD, DriveAttr: DWORD, AttrThres: DWORD
        uses    esi, edi            ;Сохранение значений регистров в стеке

; Определение размера буферов под значения атрибутов
  ATTR_SIZE  =size SendCmdOutParams +READ_ATTRIBUTE_BUFFER_SIZE
  THRES_SIZE =size SendCmdOutParams+READ_THRESHOLD_BUFFER_SIZE

; Очистка буферов для значений атрибута
        call    RtlZeroMemory, DriveAttr, size DriveAttribute
        call    RtlZeroMemory, AttrThres, size AttrThreshold

; Подготовка буфера SCIP для чтения атрибутов S.M.A.R.T.
        call    InitSMARTInBuf,  Drive,  IDE_SMART_READ_ATTRIBUTES,
```

```
offset SCIP, READ_ATTRIBUTE_BUFFER_SIZE

; Чтение значений атрибутов
        call    DeviceIoControl,    hDrive,    DFP_RECEIVE_DRIVE_DATA,
offset SCIP, size SCIP, offset SCOP, ATTR_SIZE, offset bReturned, 0
        test    eax, eax            ;Проверка успешного выполнение команды
        jz      mRSExit             ;В случае ошибки возвращается 0.

; Буфер SCOP.bOutBuffer содержит идущие друг за другом значения в
; формате структуры DriveAttribute, начиная со 2-го байта
        mov     esi, offset SCOP.bOutBuffer[2]
        mov     ecx, 30

mNextAttr:
        lodsb                       ;Загрузка элемента строки [DS:SI] в AL
        cmp     al, 194
        je      mFoundAttr
        cmp     al, 231
        je      mFoundAttr
        add     esi, size DriveAttribute-1
        loop    mNextAttr
        mov     eax, -1             ;Если атрибут не найден, возвращается -1
        jmp     mRSExit

mFoundAttr:
        dec     esi                 ;Корректировка адреса найденной
                                    ;структуры, после команды lodsb
        mov     edi, DriveAttr      ;edi = буфер для копирования данных
        mov     ecx, size DriveAttribute ;ecx = размер данных
        rep     movsb               ;Копирование данные в буфер

; Подготовка буфера SCIP для чтения пороговых значений S.M.A.R.T.
        call    InitSMARTInBuf, Drive, IDE_SMART_READ_THRESHOLDS,
offset SCIP, READ_THRESHOLD_BUFFER_SIZE

; Чтение пороговых значений
        call    DeviceIoControl,    hDrive,    DFP_RECEIVE_DRIVE_DATA,
offset SCIP, size SCIP, offset SCOP, THRES_SIZE, offset bReturned, 0
        test    eax, eax
        jz      mRSExit
```

```
        mov     esi, offset SCOP.bOutBuffer[2]
        mov     ecx, 30

mNextThres:
        lodsb
        cmp     al, 194
        je      mFoundThres
        cmp     al, 231
        je      mFoundThres
        add     esi, size AttrThreshold-1
        loop    mNextThres
        mov     eax, -1
        jmp     mRSExit

mFoundThres:
        dec     esi
        mov     edi, AttrThres
        mov     ecx, size AttrThreshold
        rep     movsb
        mov     eax, 1                          ;Возвращается 1 (успех).
mRSExit:    ret
ReadSMARTAttr     ENDP

;=== Инициализация структуры SendCmdInParams ===============
; Drive – номер диска, Feature – номер функции,
; Buffer – указатель на структуры данных, AddSize – размер буфера.
InitSMARTInBuf     PROC
        ARG Drive:DWORD, Feature:DWORD, Buffer:DWORD, AddSize:DWORD

        mov     eax, AddSize
        push    eax

; Вычисление размера буфера для чтения параметров
        add     eax, size SendCmdInParams
        call    RtlZeroMemory, Buffer, eax  ;Очистка буфера
        pop     eax                         ;eax = AddSize
        mov     SCIP.cBufferSize, eax       ;Запись размера буфера
        mov     eax, Feature                ;Запись номер функции
        mov     SCIP.irDriveRegs.bFeaturesReg, al
        mov     SCIP.irDriveRegs.bSectorCountReg, 1
        mov     SCIP.irDriveRegs.bSectorNumberReg, 1
```

```
        mov   SCIP.irDriveRegs.bCylLowReg, SMART_CYL_LOW
        mov   SCIP.irDriveRegs.bCylHighReg, SMART_CYL_HI
        mov   al, byte ptr Drive
        mov   SCIP.bDriveNumber, al
;Вычисление номера диска/головки
        and   al, 1
        shl   al, 4
        or    al, 0A0h
        mov   SCIP.irDriveRegs.bDriveHeadReg, al
        mov   SCIP.irDriveRegs.bCommandReg, IDE_COMMAND_SMART
        ret
InitSMARTInBuf     ENDP

;--- Основной код ----------------------------------------------------------
Start:
  call   printf, offset msgSMART  ;Вывод приветствия
  add    esp, 4*1                  ;Корректировка стека после print
  xor    esi, esi                  ;esi - счетчик успешно открытых устройств
  mov  ebx, 0                      ;ebx - номер текущего диска
mNextDrive:
  call   InitSMART, ebx            ;Активация S.M.A.R.T.
  test   eax, eax                  ;Проверка результата
  js     mNoDrive       ;Переход, если диск открыть не удалось (eax = -1).
  jz     mNoSMART       ;Переход, если ошибка (eax = 0).

; Чтение S.M.A.R.T. атрибутов
  call   ReadSMARTAttr, ebx, offset drvAttr, offset drvThres
  test   eax, eax          ;Проверка результата
  jz     mNoSMART  ;Переход, если ошибка (eax = 0)
  js     mNoAttr           ;Переход, если атрибут не найден (eax = -1)
  inc    esi               ;Увеличение значения счетчика открытых устройств

; Занесение в стек значений в обратном порядке для передачи параметров printf
  movzx eax, drvAttr.bRawValue[0]
  push  eax
  movzx eax, drvAttr.bRawValue[1]
  push  eax
  movzx eax, drvAttr.bRawValue[2]
  push  eax
  movzx eax, drvAttr.bRawValue[3]
  push  eax
```

```
  movzx eax, drvAttr.bRawValue[4]
  push  eax
  movzx eax, drvAttr.bRawValue[5]
  push  eax
  movzx eax, drvThres.bWarrantyThreshold
  push  eax
  movzx eax, drvAttr.bWorstValue
  push  eax
  movzx eax,drvAttr.bAttrValue
  push  eax
  movzx eax,drvAttr.bAttrID
  push  eax
  push  ebx
  call  printf, offset msgSMARTInfo  ;Вывод значений атрибута
  add   esp, 4*12

mClose:
  call  CloseHandle, hDrive          ;Закрытие устройства

mNoDrive:
  inc   ebx                  ;Увеличение номера диска
  cmp   ebx, 10              ;Сравнение его с максимальным
  jbe   mNextDrive           ;Повтор цикла, если он не превышает 10
  test  esi, esi             ;Проверка количества успешно открытых устройств
  jnz   mExit                ;Переход, если их больше 0, иначе
  call  printf, offset msgNoDrives   ;вывод сообщения об ошибке.
  add   esp, 4*1

mExit:
  call  ExitProcess, 0               ;Выход из программы

mNoAttr:                             ;В случае ошибки чтения атрибута
  call  printf, offset msgNoAttr, ebx  ;вывод сообщения об ошибке.
  add   esp, 4*2
  jmp   mClose

mNoSMART:                            ;В случае ошибки активации S.M.A.R.T.
  call  printf, offset msgNoSMART, ebx  ;вывод сообщения об ошибке.
  add   esp, 4*2
  jmp   mClose
END   Start
```

5. Требования к отчёту

Отчёт должен содержать:

– титульный лист с указанием названия ВУЗа, кафедры, номера и темы лабораторной работы, а также фамилии и инициалов студента, подготовившего отчёт;
– цель работы;
– вариант задания;
– графический алгоритм программы с кратким описанием;
– полный листинг программы SMART отредактированный согласно варианту задания;
– снимок экрана монитора с результатом работы программы;
– выводы.

6. Контрольные вопросы

6.1. Что такое S.M.A.R.T. и его атрибуты?
6.2. От чего зависит работоспособность жёсткого диска?
6.3. По каким критериям можно определить скорый выход из строя жёсткого диска?
6.4. Как можно определить сколько раз включался компьютер и сколько часов он проработал?
6.5. Как можно получить значения атрибутов?
6.6. Назначение функции DeviceIoControl и её формат.
6.7. Какова структура буфера ввода данных функции DeviceIoOControl?
6.8. Какова структура буфера вывода данных функции DeviceIoControl?
6.9. Нарисуйте графический алгоритм процедуры InitSMART программы SMART.
6.10. Как работать со структурами данных в Ассемблере?

7. Рекомендуемая литература

7.1. Мюллер, С. Модернизация и ремонт ПК, 19-е издание. [Текст]: Пер. с англ. / С. Мюллер. – М.: ООО «И.Д. Вильямс», 2011. – с.540…542.
7.2. CreateFile function (Windows) [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/en-us/library/windows/desktop/aa363858(v=vs.85).aspx, свободный – 26.04.2017.
7.3. DeviceIoControl function (Windows) [Электронный ресурс]. – Режим доступа: http://msdn.microsoft.com/en-us/library/aa363216(VS.85).aspx, свободный – 26.04.2017.
7.4. Пирогов, В.Ю. Ассемблер для Windows. [Текст] Изд. 4-е перераб. и доп. / В.Ю. Пирогов. – СПб.: БХВ-Петербург, 2011. – с. 200…207, 344…349.
7.5. Аблязов, Р.З. Программирование на ассемблере на платформе x86-64. [Текст] / Р.З. Аблязов. – М.: ДМ К Пресс, 2011. – с. 149…153.

Лабораторная работа №4
Работа с интерфейсом ввода-вывода ПЭВМ

1. Цель работы

Практическое овладение навыками работы с последовательным интерфейсом (COM-портом) ПЭВМ, используя порты ввода-вывода и средства Win API.

2. Теоретический материал

В некоторых прикладных задачах программисту приходится работать с низкоскоростными устройствами (различными микроконтроллерами, устройствами ввода-вывода аналоговой и цифровой информации). Как правило, такие устройства оснащены универсальными асинхронными приёмопередатчиками (УАПП) и общение с ними возможно через последовательный порт (COM-порт) компьютера. Далее будем рассматривать работу с УАПП 16550A и совместимые с ним.

Работать с последовательным портом можно несколькими способами:

– используя средства MS DOS в командной строке (Mode, Type и др.);

– через программные прерывания;

– через порты ввода-вывода;

– используя функции Win32 API (CreateFile, ReadFile, WriteFile, GetCommState, SetCommState, CloseHandle и др.);

– используя библиотечные функции языков программирования и программных платформ (в C++ open, read, write, close; в .NET SerialPort).

Далее рассмотрим некоторые из них подробнее.

2.1. Программные прерывания для работы с COM-портом

BIOS через **программное прерывание INT 14h** представляет программе пользователя набор стандартных функций для работы с последовательным портом (табл. 4.1). Для прерывания **INT 14h** номер функции задаётся через регистр AH, отправляемые в порт данные через AL, а номер COM-порта указывается в DX (COM1 – 0000h, COM2 – 0001h и т.д.). Если функция возвращает какие-либо данные, то она это делает через регистры AH и AL. Поскольку для задания скорости обмена во время инициализации отводится только три бита в регистре AH, то устанавливаемая скорость не превышает 9600 бод. Формат входных и выходных данных можно найти в рекомендуемой литературе [7.1].

Также с последовательным портом можно работать через программное прерывание INT 21h, но этот способ используется редко.

Таблица 4.1

Основные функции INT 14h

Код в AH	Функция	Входные регистры	Выходные регистры
00h	Инициализация COM-порта	AL – параметры инициализации, DX – номер порта	AH – состояние порта, AL – состояние модема
01h	Передача байта (символа) в порт	AL – символ, DX – номер порта	AH – состояние порта
02h	Приём байта из порта	DX – номер порта	AH – состояние порта, AL – принятый символ
03h	Получение состояния порта	DX – номер порта	AH – состояние порта, AL – состояние модема

2.2. Работа с COM-портом через порты ввода-вывода

Ещё один способ работы с последовательным портом заключается в использование адресов ***портов ввода-вывода***, назначенных COM-порту. В процессе начального тестирования POST BIOS проверяет наличие последовательных портов (регистров UART 8250 или совместимых) по стандартным адресам и помещает базовые адреса обнаруженных портов в ячейки ***BIOS Data Area*** с 0040:0000h по 0040:0007h (табл. 4.2). Нулевое значение адреса является признаком отсутствия порта с данным номером, однако в большинстве случаев эти ячейке заполнены, даже если физически COM-портов нет. Перед работой с последовательным портом рекомендуется определить его адрес путем чтения соответствующей области BIOS, а не брать стандартное значение, так как возможно назначение альтернативных адресов.

Таблица 4.2

Адреса переменных BIOS для COM-портов

Имя порта	Адрес в BIOS	Базовый адрес по умолчанию
COM1	0040:0000h – 0040:0001h	3F8h
COM2	0040:0002h – 0040:0003h	2F8h
COM3	0040:0004h – 0040:0005h	3E8h
COM4	0040:0006h – 0040:0007h	2E8h

Каждый УАПП содержит 8 специальных управляющих и контрольных регистров (портов ввода-вывода) выполняющих одну или несколько функций в зависимости от режима работы COM-порта. Эти регистры располагаются один за другим, начиная с базового адреса. Назначение регистров COM1 в зависимости от состояния седьмого

бита D7 порта 3FBh (бита DLAB – Divisor Latch Access Bit, бит загрузки делителя) приведено в табл. 4.3. Все регистры имеют размер 8 бит.

Таблица 4.3

Назначение регистров порта COM1

Адрес регистра	Состояние бита DLAB рег. 3FBh	Операция	Название регистра
3F8h	0	Запись	THR – Регистр передатчика
3F8h	0	Чтение	RBR – Регистр приёмника
3F8h	1	Запись/чтение	DLL – Регистр делителя частоты (младший байт)
3F9h	1	Запись/чтение	DIM – Регистр делителя частоты (старший байт)
3F9h	0	Запись/чтение	IER – Регистр разрешения прерываний
3FAh	x	Чтение	IIR – Регистр идентификации прерывания
3FAh	x	Запись	FCR – Регистр управления буфером FIFO
3FBh	x	Запись/чтение	LCR – Регистр управления линией
3FCh	x	Запись/чтение	MCR – Регистр управления модемом
3FDh	x	Чтение	LSR – Регистр состояния линии
3FEh	x	Чтение	MSR – Регистр состояния модема
3FFh	x	Запись/чтение	SCR – Рабочий временного хранения

Если бит загрузки делителя DLAB сброшен, т.е. установлен в 0, то регистр 3F8h может быть регистром приёмником или передатчиком в зависимости от выполняемой операции (чтение или запись данных). Регистр передатчика предназначен для временного хранения байта данных, автоматически выводимого на линию TxD, а регистр приёмника – для временного хранения вводимого байта данных с линии RxD. Если бит DLAB равен 1, то регистр 3F8h обрабатывает младший байт делителя частоты для скорости передачи данных, а регистр 3F9h – старшую часть делителя.

Необходимая скорость (V) передачи данных (в бод) задаётся значением делителя K, который определяется по формуле

$$K = \frac{115\,200}{V}.$$

Делитель имеет размер в два байта, поэтому занимает два регистра. Начиная со скорости 600 бод старшая часть делителя равна 0. В табл. 4.4 приведены некоторые значения байтов делителя, определяющие соответствующие скорости передачи данных.

Таблица 4.4

Коды значений для установки скорости передачи данных

Скорость, бод	Значение делителя		Скорость, бод	Значение делителя	
	Порт 3F9h	Порт 3F8h		Порт 3F9h	Порт 3F8h
110	04h	17h	4 800	00h	18h
150	03h	00h	7 200	00h	10h
300	01h	80h	9 600	00h	0Ch
600	00h	C0h	14 400	00h	08h
1 200	00h	60h	19 200	00h	06h
1 800	00h	40h	38 400	00h	03h
2 400	00h	30h	57 600	00h	02h
3 600	00h	20h	115 200	00h	01h

Регистр управления линией 3FBh определяет формат передаваемых данных и управляет выбором назначения портов 3F8h и 3F9h. Назначение битов регистра 3FBh приведено на рис. 4.1.

Анализируя регистр состояния линия 3FDh можно контролировать правильность обмена данными по интерфейсу RS232. Интерпретация возможных состояний битов регистра 3FDh приведена на рис. 4.2.

Обращаясь к COM-порту через порты ввода-вывода имеется возможность перевести его в диагностический режим для проверки работы без физического подключения к разъему внешних устройств. В этом режиме внутри УАПП организуется аппаратная «заглушка», соединяющая выход сдвигающего регистра передатчика со входом приёмника. Для перевода COM-порта в диагностический режим необходимо установить в 1 четвёртый бит регистра управления модемом MCR.

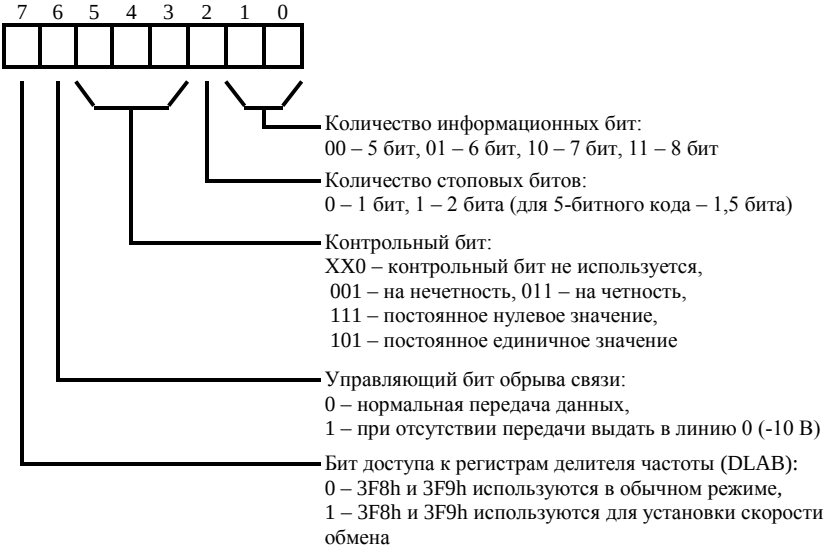

Рис. 4.1 – Назначение битов регистра 3FBh

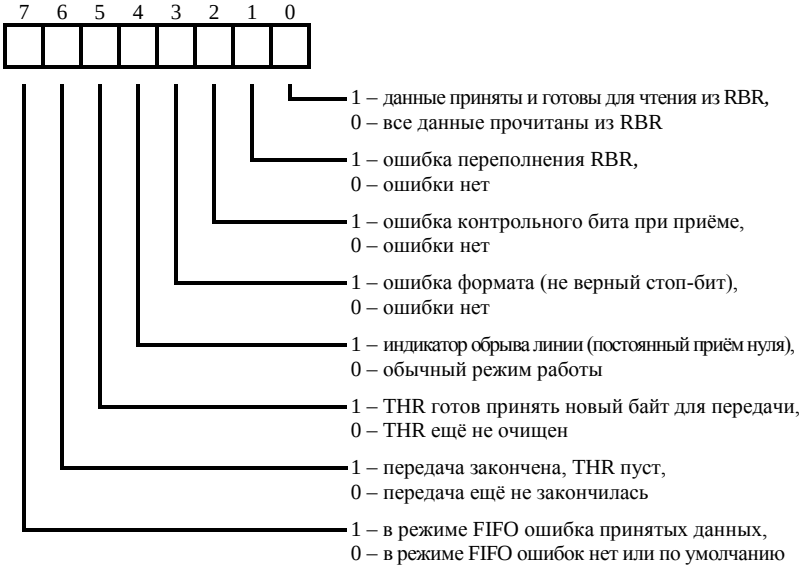

Рис. 4.2 – Состояния битов регистра 3FDh

Инициализация последовательного порта используя порты ввода-вывода включает в себя следующие шаги:

– установка в 1 седьмого бита регистра управления линией LCR;

– запись кодов делителя частоты в старший и младший регистры делителя частоты;

– задание параметров приёма-передачи с 0 в седьмом бите в регистр управления линией LCR;

– запись управляющего байта в регистр управления модемом (например, если требуется перевести порт в диагностический режим работы);

– запись управляющего байта в регистр разрешения прерываний (в большинстве случае прерывания не используются и поэтому в данный регистр во все разряды записывается 0).

Процесс работы с COM-портом состоит из нескольких этапов:

– определение наличия нужного порта;

– инициализация последовательного порта;

– передача и приём данных с контролем состояния;

– завершение работы с портом.

2.3. Работа с COM-портом используя функции Win API

В современных версиях операционной системы Windows работать напрямую с регистрами портов ввода-вывода в пользовательских приложениях запрещено, это возможно только через драйвера на уровне ядра операционной системы. Однако для доступа к COM-порту имеется большое число функций в стандартных библиотеках Win API.

В Windows для работы с коммуникационными портами используются те же функции, что и для работы с файлами. Для доступа к COM-порту необходимо получить его дескриптор функцией **CreateFile**. Функция имеет следующие параметры:

1. *lpFileName* – системное имя ресурса или имя COM-порта в формате «\\.\COMn», где n – номер порта.

2. *dwDesiredAccess* – режим доступа к ресурсу на чтение и запись.

3. *dwShareMode* – режим совместного использования. COM-порты ПК не поддерживают совместный доступ (только одна программа может открыть порт), поэтому этот параметр должен быть равен 0.

4. *lpSecurityAttributes* – параметр безопасности, который для COM-портов не используется и поэтому всегда равны 0.

5. *dwCreationDistribution* – выполняемое действие. Для COM портов должно выполняться действие открытия ресурса «OPEN_EXISTING», которое имеет значение 3h.

6. *dwFlagsAndAttributes* – задает атрибуты работы объекта. В зависимости от режима работы с последовательным портом может принимать два значения: для синхронного режима – 0, для асинхронного – FILE_FLAG_OVERLAPPED (400000000h).

7. *hTemplateFile* – дескриптор файла «шаблона». Для COM-портов не используется поэтому всегда равен 0.

Если открыть указанный COM-порт удалось, то в регистр EAX возвращается его дескриптор, а в случаях отсутствия порта с таким именем или если он занят другим приложением, то в EAX возвращается -1.

После использования дескриптор порта должен быть освобождён вызовом функции **CloseHandle** с указанием закрываемого дескриптора в качестве единственного параметра.

Перед работой с последовательным портом необходимо его настроить. Настройка порта заключается в заполнении управляющих структур и последующем вызове функций настройки Основные параметры работы порта задаются в двух управляющих структурах: *DeviceControlBlock* (или сокращённо *DCB*) и *CommTimeOuts*. Наименование и назначение полей управляющих структур представлено в табл. 4.5 и 4.6.

Четырех байтовое поле *fBitFields* структуры *DCB* содержит флаги и атрибуты работы порта. Назначение каждого бита и его значение по умолчанию можно найти в рекомендуемой литературе [7.2].

Существует несколько способов инициализации последовательных портов используя функции Win API, из которых рассмотри только один. Он заключается в использовании функций **GetCommState**, **SetCommState**, **GetCommTimeouts** и **SetCommTimeouts**. Для того, чтобы не ошибиться при заполнении полей *DCB* и *CommTimeOuts* можно получить их текущие значения с помощью **GetCommState** и **GetCommTimeouts**, а затем изменить нужные поля и записать эти структуры функциями **SetCommState** и **SetCommTimeouts**. Данные команды имеют одинаковый формат записи: первым параметром указывается *дескриптор порта*, а вторым – *адрес соответствующей структуры*, куда будет записана текущая информация или откуда будет взята информация для записи в порт. Если команда не выполнена, например, по причине не правильного указанного значения поля структуры, она вернёт нулевое значение в регистре EAX.

Таблица 4.5

Структура настроек коммуникационного устройства (DCB)

Имя поля структуры	Размер поля, байт	Описание
DCBlength	4	Размер структуры в байтах
BaudRate	4	Скорость передачи данных
fBitFields	4	Битовые поля
wReserved	2	Зарезервировано
XonLim	2	Минимальное число символов в приёмном буфере для посылки XON
XoffLim	2	Максимальное число символов в приёмном буфере перед посылкой XOFF
ByteSize	1	Число информационных бит в посылке
Parity	1	Выбор схемы контроля четности 0-4 = 0 – отсутствует бит четности; 1 – проверка нечетности; 2 – проверка четности; 3 – всегда 1; 4 – всегда 0.
StopBits	1	Количество стоповых битов: 0 = 1 стоп-бит; 1 = 1.5 стоп-бита (для 5 битовых данных); 2 = 2 стоп-бита (для 6,7,8 битовой посылки)
XonChar	1	Код XON при передаче и приёме
XoffChar	1	Код XOFF при передаче и приёме
ErrorChar	1	Код символа, заменяющий посылку при обнаружении ошибки четности
EofChar	1	Код символа конца данных
EvtChar	1	Код символа для вызова событий
wReserved1	2	Зарезервировано

Для чтения и записи данных в порт можно воспользоваться стандартными одноименными функциями для работы с файлами: **ReadFile** и **WriteFile**. Эти функции используются как для синхронного режима работы с портом, так и для асинхронного. Асинхронный режим позволяет реализовать работу по событиям, в то время как синхронный лишен этой возможности, но является более простым в реализации.

Таблица 4.6

Структура временных параметров порта (CommTimeOuts)

Имя поля структуры	Размер поля, байт	Описание
ReadIntervalTimeout	4	интервал между символами
ReadTotalTimeoutMultiplier	4	множитель для периода простоя чтения
ReadTotalTimeoutConstant	4	постоянная для периода простоя чтения
WriteTotalTimeoutMultiplier	4	множитель для периода простоя записи
WriteTotalTimeoutConstant	4	постоянная для периода простоя записи

Функции **ReadFile** и **WriteFile** имеют схожий формат и содержат 5 параметров:

1. *hFile* – дескриптор порта;
2. *lpBuffer* – адрес буфера, куда будут записываться данные из порта или откуда будут данные отправляются в порт;
3. *nNumberOfBytesToRead* или *nNumberOfBytesToWrite* – число получаемых и записываемых байт;
4. *lpNumberOfBytesRead* или *lpNumberOfBytesWritten* – указатель на переменную, которая будет содержать число полученных или отправленных в порт байт после завершения операции;
5. lpOverlapped – указатель на структуру OVERLAPPED.

Последний параметр передаёт настройки для асинхронного чтения или записи данных. Если порт был открыт с параметром FILE_FLAG_OVERLAPPED, этот параметр должен указывать на структуру типа *OVERLAPPED*, если же порт был открыт для работы в синхронном режиме, то этот указатель обязательно должен быть 0.

Функции возвращают в EAX ненулевые значения, если прочитано или записано необходимое количество байтов. В случае ошибки возвращается 0.

Для контроля записи данных в порт можно сравнить 3 и 4 параметр функции после её завершения.

При синхронном чтении данных с порта приходится циклически проверять их наличие. Для этого можно использовать поле *cbInQue* структуры *ComStat* и функцию **ClearCommError**. Структура *ComStat* содержит информацию о текущем состоянии коммуникационного устройства. Назначение её полей представлено в табл.4.7.

Функция **ClearCommError** получает информацию об ошибках порта и сбрасывает флаги ошибок. Функция содержит три параметра:

дескриптор порта, указатель на переменную, куда будет записан код ошибки и указатель на структуру *ComStat*. Битовую маску кода ошибки можно найти в рекомендуемой литературе [7.3].

Таблица 4.7

Структура с информацией о коммуникационном устройстве (ComStat)

Имя поля структуры	Размер поля, байт	Описание
ComStateFlags	4	Флаги состояний линий порта
cbInQue	2	Число байт полученных драйвером, но ещё не прочитанных **ReadFile**
cbOutQue	2	Число байт переданных функцией **WriteFile** драйверу, но ещё не отправленные в УАПП

Перед началом и завершением работы с коммуникационным портом желательно очисть входной и выходной буферы указанного ресурса. Для этого предназначена функция **PurgeComm**. Функция имеет два параметра: дескриптор порта и атрибут *dwFlags*, задающий код выполняемой операции. Битовая маска кода выполняемой операции представлена на рисунке 4.3.

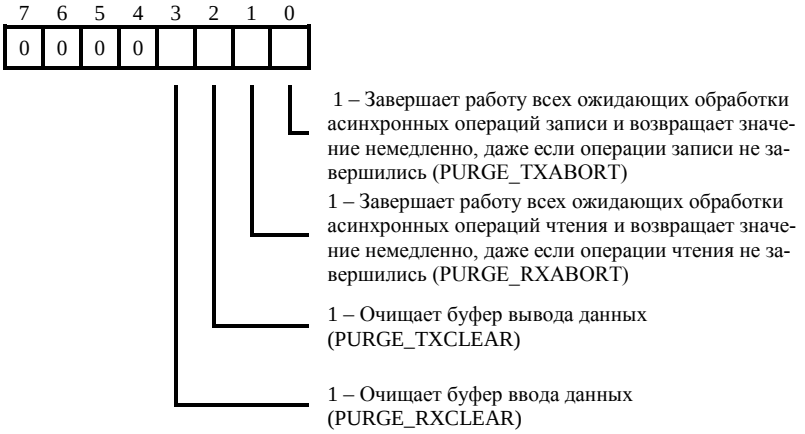

Рис. 4.3 – Битовая маска параметра dwFlags функции PurgeComm

Как уже упоминалось ранее в случае невыполнения функции Win API или выполнении её с ошибкой (не путать с ошибками коммуникационного порта) в регистр EAX записывается 0. Для идентифика-

ции ошибки можно воспользоваться функцией **GetLastError**, которая указывается без параметров и возвращает код последней ошибки. Коды ошибок для каждой функции разные и их можно найти в [7.3].

Пример программы, работающей с COM-портом через функции библиотек Win API, рассмотрен в разделе 2.5.

2.4. Программа обращения к COM-порту через команды ввода-вывода

Далее приведён листинг программы на Ассемблере, которая передаёт код символа, введённого с клавиатуры, в первый COM-порт и получает эхо сигнал (в диагностическом режиме работы порта), который выводится на экран [7.4]. COM-порт инициализирован на скорость 600 бод, 8 бит данных, без контроля чётности, с 1 стоповым битом. Данные вводятся с использованием программного прерывания INT 16h и выводятся на экран с помощью программного прерывания INT 10h. Выход из программы по нажатию клавиши ESC.

Программа является 16-ти разрядной, поэтому для её трансляции необходимо использовать 16-ти разрядный пакет **TASM**. Поскольку по умолчанию в BIOS заданы адреса двух COM-портов, данная программа может запускаться даже на компьютерах где нет физических COM-портов. На 64-х битных версиях операционной системы Windows программу можно запустить через **DOSBox**.

```
TITLE COM-PORT
;16-ти разрядное приложение проверки работы COM-порта.
;Программа передаёт код символа, введённого с клавиатуры, в первый
;COM-порт и получает эхо сигнал, который выводит на экран.

.MODEL SMALL
.STACK 100h

;--- Данные ------------------------------------------------------------------
.DATA
  msgTitle     db 'Testirovanie posledovatelnogo porta', 13, 10, 13, 10, '$'
  msgInfo      db 'Nazhmite lyubuyu klavishu dlya otpravki yeyo koda v port.'
               db 13, 10, 'Dlya vykhoda nazhmite ESC.', 13, 10, 13, 10, '$'
  msgInit      db 'Com-port inizializirovan', 13, 10, 13, 10, '$'
  msgError     db 'Port ne obnaruzhen', 13, 10, '$'

.CODE
;--- Очистка экрана и установка курсора в верхний левый угол-----------
ClearScreen PROC
```

```
        mov     ah, 06h         ;функция прокрутки активной страницы вверх
        mov     al, 00h         ;очистка всего экрана
        mov     bh, 07h         ;атрибут пробела = Ч/Б, норм. яркости
        mov     cx, 00h         ;верхняя левая позиция
        mov     dx, 184Fh       ;нижняя правая позиция (Y=24, X=79)
        int     10h
        mov     ah, 02h         ;установка курсора
        mov     bh, 00h
        mov     dx, 00h
        int     10h
        ret
ClearScreen ENDP

;--- Вывод текста на экран ----------------------------------------------------
Print PROC
        push    ax
        mov     ah, 9
        int     21h
        pop     ax
        ret
Print ENDP

;--- Инициализация C0M-порта---------------------------------------------------
;DX – базовый адрес порта COM-порта
InitCom PROC
        push    dx
        add     dx, 03h         ;регистр управления линии (03FBh)
        mov     al, 10000000b   ;устанавливаем 7 бит
        out     dx, al
        sub     dx, 2           ;старший байт регистра делителя частоты (3F9h)
        mov     al, 00h         ;код для 600 бод
        out     dx, al
        dec     dx              ;младший байт регистра делителя частоты (3F8h)
        mov     al, 0C0h        ;код для 600 бод
        out     dx, al
        add     dx, 03h         ;регистр управления линией (3FBh)
        mov     al, 00h         ;очищаем al
        or      al, 00000011b   ;длина символа (11b) – 8 бит данных
        or      al, 00000000b   ;длина стоп-бита (0b) – 1 стоп-бит
        or      al, 00000000b   ;наличие бита чётности (000b) –
                                ;не генерировать бит четности
```

```
        out     dx, al
        inc     dx              ;регистр управления модемом (3FCh)
        mov     al, 00010000b   ;диагностический режим (0001XXXXb)
        out     dx, al
        sub     dx, 3           ;регистр разрешения прерываний (3F9h)
        mov     al, 0           ;прерывания запрещены
        out     dx, al
        pop     dx
        ret
InitCom ENDP

;--- Преобразование кода символа для вывода в бинарном виде ---------
;AX – ASCII код символа
Preobr PROC
        mov     bl, al
        mov     ah, 0Eh
        mov     cx, 8
m6:     rol     bl, 1
        jc      m7
        mov     al, 30h         ;вывод символа "0"
        jmp     m8
m7:     mov     al, 31h         ;вывод символа "1"
m8:     int     10h
        loop    m6
        ret
Preobr ENDP

;--- Работа с портом -------------------------------------------------------
;DX – базовый адрес COM-порта
Work PROC
m1:     mov     ah, 00h         ;ввод символа
        int     16h             ;AL=ASCII код, AH=сканкод
        mov     ah, 00h         ;сканкод удаляем
        cmp     al, 1Bh         ;проверка нажатия ESC
        je      m5
        push    ax
        push    ax
        mov     ah, 0Eh         ;вывод этого символа на экран
        int     10h
        mov     al, 3Dh         ;вывод символа "="
        int     10h
```

```
        pop     ax
        call    Preobr
        mov     al, 1Ah         ;вывод символа "стрелочка"
        int     10h
        add     dx, 05h         ;регистр состояния линии (3FDh)
        mov     cx, 0Ah         ;счётчик цикла равен 10
m2:     in      al, dx          ;забираем из порта 03FDh данные в AL
        test    al, 20h         ;готов к приему данных? (00100000b)
        jz      m3
        loop    m2
m3:     sub     dx, 05h         ;регистр передатчика (03F8h)
        pop     ax              ;загрузить передаваемый байт из стека
        out     dx, al          ;передаём в COM-порт код символа из AL
        add     dx, 05h         ;регистр состояния линии (03FDh)
m4:     in      al, dx          ;данные приняты ?
        test    al, 01h
        jz      m4
        sub     dx, 05h         ;регистр приёмника (03F8h)
        in      al, dx          ;прочитать принятые данные
        mov     ah, 0
        push    ax
        mov     ah, 0Eh         ;вывести символ на экран
        int     10h
        mov     al, 3Dh         ;вывод символа "="
        int     10h
        pop     ax
        call    Preobr
        mov     al, 13          ;перевод в начало строки
        int     10h
        mov     al, 10          ;переход на другую строчку
        int     10h
        jmp     m1
m5:     ret
Work ENDP

;--- Основной код ----------------------------------------------------------------
Start:  mov     ax, @Data
        mov     ds, ax
        call    ClearScreen     ;очистка экрана
        lea     dx, msgTitle
        call    Print           ;вывод строки msgTitle
```

```
        mov   ax, 40h          ;ES указывает на область данных BIOS
        mov   es, ax
        mov   dx, es:[0]       ;получаем базовый адрес COM1 (03F8h)
        test  dx, 0FFFFh       ;адрес порта есть?
        jnz   m0
        lea   dx, msgError
        call  Print
        jmp   mExit
m0:     call  InitCom          ;инициализация COM-порта
        push  dx
        lea   dx, msgInit
        call  Print
        lea   dx, msgInfo
        call  Print
        pop   dx
        call  Work             ;работа с портом
mExit:  mov   ax, 4C00h
        int   21h
END Start
```

2.5. Программа обращения к COM-порту через Win API

Пример программы на Ассемблере передающей данные в Com-порт используя функции Win API представлен ниже. В программе инициализируется первый COM-порт (на компьютере имеется физический порт) на скорости 600 бод, 8 бит данных, без контроля чётности, с 1 стоповым битом, параметры тайм-аута для операции записи задаются как 100 и 10. Программа создаёт новое консольное окно для вывода приветствия и ввода кода нажатой клавиши для передачи его в порт.

Для получения кода символа используется низкоуровневая функция ввода-вывода **ReadConsoleInput**. Данная функция имеет следующие параметры:

1. *hConsoleInput* – дескриптор входного буфера консоли;
2. *lpBuffer* – указатель на массив структур буфера событий консоли;
3. *nLength* – число получаемых структур в буфере событий консоли;
4. *lpNumberOfEventsRead* – указатель на переменную, которая получает количество прочитанных структур.

Массив структур, возвращаемый данной функцией, содержит информацию обо всех событиях, происходящих с консольным окном (табл. 4.8). Поскольку необходимо получить только код нажатой клавиши структура входного буфера инициализируется упрощённо и формат каждой структуры не рассматривается.

Таблица 4.8

Состав буфера событий консольного окна

Имя структуры	Имя поля структуры	Размер поля, байт	Значение поля	Описание
Input-Record	EventType	2		тип события
	KeyEventRecord		KeyEvent	структура с событиями от клавиатуры
	MouseEvent-Record	16		структура с событиями от мыши
	WindowsBuffer-SizeRecord	4		структура с новыми размерами экранного буфера
	MenuEvent-Record	4		структура с внутренними событиями ОС
	FocusEvent-Record	4		структура с внутренними событиями ОС
KeyEvent	KeyDown	4		признак нажатия (>0) или отпускания (<=0) клавиши
	RepeatCount	2		количество повторов при удержании клавиши
	VirtualKeyCode	2		виртуальный код клавиши
	VirtualScanCode	2		скан-код клавиши
	ASCIIChar	2		ASCII-код клавиши
	ControlKeyState	4		состояние управляющих клавиш

Полученный ASCII-код нажатой клавиши выводится в символьном и двоичном виде в консоль, а также отправляется в порт. После проверки успешной отправки данных УАПП выводится соответствующее сообщение. В случае обнаружения ошибки её код выводится в консоль. Выход из программы реализован по нажатию клавиши Esc.

Исходный модуль программы сохраняется в кодировке OEM866. Программа создаётся с помощью 32-х битного пакета **TASM32** с ключами:

tasm32 /ml /z /zi /n <имя файла>
tlink32 /Tpe /ap /x /v <имя файла>.

В случае отсутствия порта COM1 на запускаемом компьютере выводится сообщение об ошибке доступа к ресурсу.

```
TITLE COM-PORT32
;32-х разрядное приложение для отправки данных в COM-порт.
;Программа передаёт код символа, введённого с клавиатуры, в первый
;COM-порт, работающий в синхронном режиме.

.386
;Плоская модель памяти и стандартная модель вызова подпрограмм.
.MODEL FLAT, STDCALL

;Директивы компоновщику для подключения библиотек.
INCLUDELIB import32.lib          ; Работа с библиотекой ОС Kernel32.

;--- Внешние WinAPI-функций ---------------------------------------------
EXTRN FreeConsole:        PROC  ; Освободить текущую консоль
EXTRN AllocConsole:       PROC  ; Создать свою консоль
EXTRN GetStdHandle:       PROC  ; Получить дескриптор текущей консоли
EXTRN printf:             PROC  ; Вывести по шаблону
EXTRN CreateFileA:        PROC  ; Получить дескриптор ресурса
EXTRN ReadConsoleInputA: PROC  ; Получить события консоли
EXTRN WriteFile:          PROC  ; Передать данные в порт
EXTRN GetLastError:       PROC  ; Получить код ошибки
EXTRN PurgeComm:          PROC  ; Очистить очередь приёма и передачи
EXTRN CloseHandle:        PROC  ; Закрыть дескриптор
EXTRN ExitProcess:        PROC  ; Завершить процесс
EXTRN GetCommState:       PROC  ; Получить DCB структуру порта
EXTRN SetCommState:       PROC  ; Задать параметры порта
EXTRN GetCommTimeouts:  PROC  ; Получить временные параметры
EXTRN SetCommTimeouts: PROC  ; Установить временные параметры

;--- Константы ----------------------------------------------------------
GENERIC_READ    = 80000000h    ; допускается чтение
GENERIC_WRITE   = 40000000h    ; допускается запись
OPEN_EXISTING           = 3    ; предписывает открывать устройство,
                               ;если оно существует
STD_INPUT_HANDLE    = -10      ; консольное окно для ввода данных
STD_OUTPUT_HANDLE = -11        ; консольное окно для вывода данных
PURGE_TXCLEAR           = 4    ; очистка очереди передачи
CBR_600                 = 600  ; значение скорости передачи данных
Key_Event               = 1    ; тип клавиатурного события
```

```
;--- Структуры данных ----------------------------------------------------------
;Структура основных параметров последовательного порта
DeviceControlBlock STRUC
        DCBlength    dd    ?   ; Размер структуры в байтах
        BaudRate     dd    ?   ; Скорость передачи данных
        fBitFields   dd    ?   ; Битовые поля
        wReserved    dw    ?   ; Зарезервировано
        XonLim       dw    ?   ; Мин. кол-во символов для посылки XON
        XoffLim      dw    ?   ; Макс. кол-во символов для посылки XOFF
        ByteSize     db    ?   ; Число информационных бит
        Parity       db    ?   ; Выбор схемы контроля четности 0-4
        StopBits     db    ?   ; Кол-во стоповых битов 0, 1, 2 = 1, 1.5, 2
        XonChar      db    ?   ; Код XON при передаче и приёме
        XoffChar     db    ?   ; Код XOFF при передаче и приёме
        ErrorChar    db    ?   ; Код символа при обнаружении ошибки
        EofChar      db    ?   ; Код символа конца данных
        EvtChar      db    ?   ; Код символа для вызова событий
        wReserved1   dw    ?   ; Зарезервировано
DeviceControlBlock ENDS

;Структура временных параметров порта
CommTimeOuts      STRUC
  ReadIntervalTimeout          dd ?  ; Интервал между символами
  ReadTotalTimeoutMultiplier   dd ?  ; Множ. для периода простоя чтения
  ReadTotalTimeoutConstant     dd ?  ; Постоян. для периода простоя чтения
  WriteTotalTimeoutMultiplier  dd ?  ; Множ. для периода простоя записи
  WriteTotalTimeoutConstant    dd ?  ; Постоян. для периода простоя записи
CommTimeOuts      ENDS

; Структура с информацией о событиях от клавиатуры
KeyEvent          STRUC
        KeyDown          dd ?    ; Признак нажатия/отпускания клавиши
        RepeatCount      dw ?    ; Кол-во повторов при удержании клавиши
        VirtualKeyCode   dw ?    ; Виртуальный код клавиши
        VirtualScanCode  dw ?    ; Скан-код клавиши
        ASCIIChar        dw ?    ; ASCII-код клавиши
        ControlKeyState  dd ?    ; Состояние управляющих клавиш
KeyEvent          ENDS

; Структура буфера событий консоли
InputRecord       STRUC
```

```
    EventType                   dw ?  ; Тип события
                                dw 0  ; для выравнивания поля
    KeyEventRecord    KeyEvent <>
    MouseEventRecord  dd 4 dup (?)
    WindowsBufferSizeRecord dd ?
    MenuEventRecord           dd ?
    FocusEventRecord          dd ?
InputRecord           ENDS

;--- Данные ------------------------------------------------------------------------
.DATA
  msgTitle     db 'Тестирование последовательного порта', 13,10,13,10, 0
  msgInfo      db 'Нажмите любую клавишу для отправки её кода в порт.'
               db 13, 10, 'Для выхода нажмите ESC.', 13, 10, 13, 10, 0
  msgInit      db 'Com-порт инициализирован', 13, 10, 13, 10, 0
  msgError     db 13, 10, 'Ошибка: %u', 13, 10, 0
  msgInKey     db '%c=', 0
  msgInKeyBin        db '       ', 0
  msgOutKey          db 1Ah, 'Данные отправлены в порт', 13, 10, 0
  msgNotOutKey       db 7Eh, 'Данные в порт не отправлены', 13, 10, 0

  NamePort     db '\\.\COM1', 0 ; Имя порта в Windows
  HPort        dd ?             ; Дескриптор порта
  HIn          dd ?             ; Дескриптор окна ввода данных
  Buffer       dw ?             ; Буфер для передачи данных
  NumEvent     dd ?             ; Число событий
  LenBuf       dd 1             ; Количество отправляемых байт в порт
  WrBuf        dd ?             ; Количество отправленных байт в порт

  DCB DeviceControlBlock <> ; Указатель на структуру с параметрами порта
  CTO CommTimeOuts       <> ; Указатель на структуру с времен. параметрами
  CBuffer InputRecord    <>  ; Указатель на структуру буфера консоли

.CODE
;--- Инициализация COM-порта -----------------------------------------------
; Возвращает EAX <> 0 - успешное завершение, 0 - ошибка
InitCom PROC
        ARG   Port: DWORD ; Дескриптор порта

; Получение текущих значений DCB структуры
        call    GetCommState, Port, offset DCB
```

```
        test    eax, eax        ; Проверка на ошибки
        jz      mICExit

; Заполнение структуры DCB
        mov     DCB.BaudRate, CBR_600
        mov     DCB.ByteSize, 8
        mov     DCB.Parity, 0
        mov     DCB.StopBits, 0

; Установка параметров порта
        call    SetCommState, Port, offset DCB
        test    eax, eax
        jz      mICExit

; Получение текущих значений временных параметров
        call    GetCommTimeouts, Port, offset CTO
        test    eax, eax
        jz      mICExit

; Заполнение структуры CTO
        mov     CTO.WriteTotalTimeoutMultiplier, 100;
        mov     CTO.WriteTotalTimeoutConstant,   10;

; Установка временных параметров порта
        call    SetCommTimeouts, Port, offset CTO
        test    eax, eax
        jz      mICExit

; Очистка буфера передачи перед работой с портом
        call    PurgeComm, Port, PURGE_TXCLEAR
mICExit:
        ret
InitCom ENDP

;--- Преобразование числа к двоичному виду --------------------------------
;Number - исходное число
;String - адрес строки, куда записывается результат преобразования
Preobr PROC
        ARG   Number:DWORD, String:DWORD
        uses    esi, ebx, ecx
```

```
        mov     esi, String
        mov     ecx, 8          ; Преобразование только младшего байта
        add     esi, ecx        ; Обратный порядок вывода: от младшего
                                ;разряда к старшему
        sub     esi, 1
@1:     shr     Number, 1
        jc      @2
        mov     bl, 30h
        jmp     @3
@2:     mov     bl, 31h
@3:     mov     [esi], bl
        dec     esi
        loop    @1
        ret
Preobr ENDP

;--- Основной код ----------------------------------------------------------
Start:
        call    FreeConsole
        call    AllocConsole
        call    GetStdHandle, STD_INPUT_HANDLE
        mov     HIn, eax
        call    printf, offset msgTitle
        add     esp, 4*1        ; Корректировка стека после printf

; Открытие порта для синхронного режима работы
        call    CreateFileA, offset NamePort, GENERIC_READ or
GENERIC_WRITE, 0, 0, OPEN_EXISTING, 0, 0
        mov     HPort, eax
        cmp     HPort, -1       ; В случае ошибки вывод сообщения
        je      mError

; Инициализация COM-порта
        call    InitCom, HPort
        test    eax, eax
        jz      mError

        call    printf, offset msgInit
        add     esp, 4*1
        call    printf, offset msgInfo
        add     esp, 4*1
```

```
; Получение ASCII-кода нажатой клавиши
cycle:
        call    ReadConsoleInputA, HIn, offset CBuffer, 1, offset NumEvent
        cmp     CBuffer.EventType, Key_Event
        jne     cycle           ; Событие не от клавиатуры?
        cmp     CBuffer.KeyEventRecord.KeyDown, 0
        jz      cycle           ; Клавиша отпущена или нажата?
        cmp     CBuffer.KeyEventRecord.ASCIIChar, 27
        je      mClose          ; Нажата клавиша Esc?

        xor     eax, eax
        mov     ax, CBuffer.KeyEventRecord.ASCIIChar
        mov     Buffer, ax

; Вывод символа и его двоичного кода в консоль
        call    Preobr, eax, offset msgInKeyBin
        call    printf, offset msgInKey, Buffer
        add     esp, 4*2
        call    printf, offset msgInKeyBin
        add     esp, 4*1

; Передача одного байта из буфера в порт
        call    WriteFile, HPort, offset Buffer, LenBuf, offset WrBuf, 0
        test    eax, eax
        jz      mError
; Проверка соответствия количества отправленных байт
        mov     eax, WrBuf
        cmp     eax, LenBuf
        jne     mNotOut
        lea     edi, msgOutKey
        jmp     mOut
mNotOut:
        lea     edi, msgNotOutKey
mOut:
        call    printf, edi
        add     esp, 4*1
        jmp     cycle

mError:                         ; Обработка ошибок
        call    GetLastError
```

```
        call    printf, offset msgError, eax
        add     esp, 4*2
        cmp     HPort, -1
        je      mWait
        call    CloseHandle, HPort

mWait:                          ; Ожидание нажатия клавиши
        call    ReadConsoleInputA, HIn, offset CBuffer, 1, offset NumEvent
        cmp     CBuffer.EventType, Key_Event
        jne     mWait           ; Событие от клавиатуры?
        jmp     mExit

; Очистка буфера передачи и закрытие порта
mClose:
        call    PurgeComm, HPort, PURGE_TXCLEAR
        test    eax, eax
        jz      mError
        call    CloseHandle, HPort

mExit:                          ; Выход из программы
        call    CloseHandle, HIn
        call    FreeConsole
        call    ExitProcess, 0
END Start
```

2.6. Утилиты для работы с COM-портом

На данный момент существует большое количество утилит, которые помогают при изучении, настройке и разработке программных и аппаратных компонентов различных систем, работающих с COM-портом. Эти утилиты можно разделить на несколько групп:

1) терминальные программы;
2) виртуальные порты и их соединения;
3) анализаторы трафика;
4) прочие специализированные программы.

Терминальные программы, работающие с последовательным портом, как правило, имеют два поля: ввода данных в порт и вывода данных с порта. С помощью таких программ можно отправить как отдельные байты, так и файлы в порт. Полученные данные с порта выводятся либо в виде чисел, либо в символьной кодировке. Некоторые программы позволяют через элементы графического интерфейса управлять служебными сигналами порта (RTS, DTR) и показывать состояние сигнальных входов порта.

Отсутствие физического последовательного порта не является сейчас препятствием для изучения принципов работы COM-порта и разработки простых приложений. Существуют программы, которые устанавливают в систему свои драйвера, тем самым создавая виртуальные устройства. При этом само виртуальное устройство или отправляет (принимает) данные в сеть TCP/IP или соединено виртуальным каналом связи с другими таким же виртуальным портом, что позволяет на одном компьютере через один порт отправлять данные, а через другой принимать их обратно. Примером такой программы является Нуль-модемный эмулятор **Com0com** [7.5].

Анализаторы трафика позволяют «прослушивать» передачу данных между программным обеспечением и аппаратным устройством, подключенным к ПК через последовательный порт. В некоторых случаях анализаторы позволяют «прослушивать» трафик между двумя устройствами, если компьютер с двумя COM-портами подключается в разрыв интерфейсной линии.

Некоторые программы совмещают в себе сразу несколько функций, например, **AccessPort** от SUDT Studio [7.6], которая позволяет одновременно в одном окне вести мониторинг передаваемых данных в порт, а в другом выводить и получать данные через другой последовательный порт.

3. Подготовка к работе

3.1. Изучить методические указания и рекомендованную литературу.

3.2. Подготовить ответы на контрольные вопросы.

4. Задание на выполнение работы

4.1. Используя текстовый редактор, создать и отредактировать исходный модуль программы **Comport.asm**, выполняющую тестирование последовательного порта. Параметры инициализации выбрать согласно варианту задания. Вариант задания определяется по последней цифре зачетной книжки *N* и берётся из табл. 4.9. Для нечётных номеров *N* программа должна работать с COM1, для четных *N* – COM2.

4.2. Запустить созданную программу **Comport.exe** и выполнить передачу нескольких символов в порт. Сделать снимок экрана монитора с результатами работы программы.

4.3. В Диспетчере устройств ОС проверить наличие и определить номера виртуальных COM-портов. В ветке *Порты (COM и LPT)* должны быть два порта, например, *com0com – serial port emulator (COM3)* и *com0com – serial port emulator (COM4)*. В случае отсутствия таких портов установить с настройками по умолчанию программу **com0com** [7.5].

Таблица 4.9

Варианты заданий

№ варианта	Скорость обмена, бод	Число бит данных	Контрольный бит	Число стоповых битов
	300	5	не используется	1
1	1 200	6	четность	2
2	2 400	7	нечетность	1
3	3 600	8	постоянно 0	2
4	4 800	5	постоянно 1	1,5
5	9 600	6	не используется	2
6	19 200	7	четность	1
7	38 400	8	нечетность	2
8	57 600	5	постоянно 0	1
9	115 200	6	постоянно 1	2

4.4. Используя текстовый редактор, создать и отредактировать исходный модуль программы **Comport32.asm**, выполняющую передачу кода нажатой клавиши в последовательный порт. Параметры инициализации выбрать согласно варианту задания из табл. 4.9. В качестве порта использовать первый по списку виртуальный порт.

4.5. Запустить программу **AccessPort** и настроить терминал на второй виртуальный порт. Запустить режим чтения порта.

4.6. Запустить созданную программу **Comport32.exe** и выполнить передачу нескольких символов в порт. Сделать снимок экрана монитора с результатами работы программы (консольное окно и окно программы **AccessPort**).

4.7. Используя текстовый редактор, создать программу **CPortRead32.exe**, выполняющую побайтное чтение данных с порта и вывода их в консольное окно. Завершение программы по нажатию Esc. Параметры инициализации выбрать такими же как в программе передачи данных в порт.

4.8. Проверить работу созданной программы, используя **AccessPort**. Сделать снимок экрана монитора с результатами работы программы (консольное окно и окно программы **AccessPort**).

4.9. Выключить режим чтения порта в программе **AccessPort**.

5. Требования к отчёту

Отчёт должен содержать:

– титульный лист с указанием названия ВУЗа, кафедры, номера и темы лабораторной работы, а также фамилии И.О. студента, подготовившего отчёт;

– цель работы;
– номер варианта и параметры инициализации COM-порта;
– листинги программ **Comport**, **Comport32**, **CPortRead32** с комментариями;
– снимки экрана монитора с результатами работы программ **Comport**, **Comport32**, **CPortRead32**.

6. Контрольные вопросы

6.1. В чём заключается инициализация COM-порта?
6.2. Приведите несколько способов инициализацию COM-порта.
6.3. Какие программные прерывания используются для работы с COM-портом?
6.4. Как задаётся скорость передачи данных при работе через порты ввода-вывода?
6.5. Как определить базовый адрес порта ввода-вывода для COM2?
6.6. Как задаётся скорость передачи данных COM-порта при использовании портов ввода-вывода?
6.7. На каких скоростях можно передавать данные через коммуникационный порт?
6.8. Для чего нужен контрольный бит и какие значения он может принимать?
6.9. Какую информацию можно получить из анализа регистра состояния линии COM-порта?
6.10. Могут ли два приложения одновременно работать с одним COM-портом?
6.11. Какой общий алгоритм работы с последовательным портом в приложениях?
6.12. В каких режимах можно работать с COM-портом используя Win API и в чём их отличия?
6.13. Что содержит структура DCB последовательного ресурса?
6.14. Что делает функция PurgeComm?
6.15. В чём отличия функций ClearCommError от GetLastError?
6.16. Приведите примеры терминальных программ для работы с COM-портом.
6.17. Как получить данные введённые с клавиатуры используя функции Win API?

7. Рекомендуемая литература

7.1. Яшкардин, В. Программирование COM порта ПК [Электронный документ] / В. Яшкардин – Режим доступа: http://www.softelectro.ru/rs232prog.html, свободный – 2.11.2017.

7.2. Магда, Ю.С. Программирование последовательных интерфейсов [Текст] / Ю.С. Магда. – СПб: БХВ-Петербург, 2009. – с. 125-159.
7.3. Communications Functions (Windows) [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa363194(v=vs.85).aspx, свободный – 2.11.2017.
7.4. Авдеев, В.А. Периферийные устройства: интерфейсы, схемотехника, программирование [Текст] / В.А. Авдеев. – М.: ДМК Пресс, 2009, с.315-323.
7.5. Null-modem emulator (com0com) – virtual serial port driver for Windows [Электронный ресурс] – Режим доступа: http://com0com.sourceforge.net/, свободный – 2.11.2017.
7.6. SUDT AccessPort Debugger for RS232/422/485 Serial Port [Электронный ресурс] / SUDT Studio – Режим доступа: http://www.sudt.com/en/ap/index.html, свободный – 2.11.2017.

Лабораторная работа №5
Изучение интерфейса USB

1. Цель работы

Получение практических навыков идентификации USB устройств и определения их режимов работы.

2. Теоретический материал

2.1. Дескрипторы USB устройств

Для взаимодействия с хостом каждое USB устройство имеет в своём составе ряд регистров, в которых хранится информация о работе устройства. Обмен информацией из этих регистров с хостом осуществляется путём стандартных запросов и получения ответов. Такие ответы оформлены в виде структурированных данных называемых **дескрипторами**. Среди всех возможных дескрипторов, которые могут отправлять устройства, рассмотрим только основные:

– **дескриптор устройства** (device descriptor) – содержит основную информацию о USB устройстве в целом и о количестве существующих конфигурациях. USB устройство может иметь только один такой дескриптор. Названия полей и их описания для стандартного дескриптора устройства показаны в табл. 5.1 [7.1].

– **уточняющий дескриптор устройства** (device qualifier descriptor) – содержит дополнительную информацию о HS-устройстве при его работе на другой скорости. Например, если устройство работает в FS-режиме, то уточняющий дескриптор вернет информацию об HS-режиме работы и наоборот. Названия полей и их описания для уточняющего дескриптора показаны в табл. 5.2.

– **дескриптор конфигурации** (configuration descriptor) – содержит информацию об одной из возможных конфигураций USB-устройства. Устройство может иметь несколько таких дескрипторов. Названия полей и их описания для дескриптора конфигурации показаны в табл. 5.3. Установленный в 1 шестой бит поля *bmAttributes* свидетельствует о наличии собственного источника питания у устройства. Установленный в 1 пятый бит того же поля является признаком возможности пробуждения устройства по внешнему сигналу. Значение поля *MaxPower* равно максимальному току в миллиамперах, потребляемому устройством от шины, умноженному на 2.

– **дескриптор интерфейса** (interface descriptor) – содержит информацию об одном из интерфейсов, доступных при определенной конфигурации USB устройства. Под интерфейсом в данном случае понимается набор функций реализуемых устройством, например, мультимедийная клавиатура может иметь два интерфейса: первый для

реализации стандартной работы клавиатуры; второй для поддержки мультимедийных клавиш. Структура дескриптора интерфейса приведена в табл. 5.4.

Таблица 5.1

Поля стандартного дескриптора устройства

Поле	Описание
bLength	Размер дескриптора в байтах
bDescriptorType	Тип дескриптора (01h)
bcdUSB	Номер версии спецификации USB в формате BCD
bDeviceClass	Код класса USB
bDeviceSubClass	Код подкласса USB устройства
bDeviceProtocol	Код протокола USB
bMaxPacketSize0	Максимальный размер пакета для нулевой конечной точки
idVendor	Идентификатор производителя
idProduct	Идентификатор продукта
bcdDevice	Номер версии устройства в формате BCD
iManufacturer	Индекс дескриптора строки, описывающего производителя
iProduct	Индекс дескриптора строки, описывающего продукт
iSerialNumber	Индекс дескриптора строки, содержащей серийный номер USB устройства
bNumConfigurations	Количество возможных конфигураций USB устройства

Таблица 5.2

Поля уточняющего дескриптора устройства

Поле	Описание
bLength	Размер дескриптора в байтах
bDescriptorType	Тип дескриптора (06h)
bcdUSB	Номер версии спецификации USB в формате BCD (равно 0200h или 0101h)
bDeviceClass	Код класса USB
bDeviceSubClass	Код подкласса USB-устройства
bDeviceProtocol	Код протокола USB
bMaxPacketSize0	Максимальный размер пакета для нулевой конечной точки
bNumConfigurations	Количество дополнительных конфигураций устройства
bReserved	Зарезервировано, должно быть равно нулю

Таблица 5.3

Поля дескриптора конфигурации

Поле	Описание
bLength	Размер дескриптора в байтах
bDescriptorType	Тип дескриптора (02h)
wTotalLength	Общий объем данных (в байтах), возвращаемый для данной конфигурации
bNumInterfaces	Количество интерфейсов, поддерживаемых данной конфигурацией
bConfigurationValue	Идентификатор конфигурации, описываемой данным дескриптором
iConfiguration	Индекс дескриптора строки, описывающей данную конфигурацию
bmAttributes	Характеристики конфигурации
MaxPower	Код мощности, потребляемой USB-устройством от шины

Таблица 5.4

Поля дескриптора интерфейса

Поле	Описание
bLength	Размер дескриптора в байтах
bDescriptorType	Тип дескриптора (04h)
bInterfaceNumber	Номер данного интерфейса (нумеруются с 0) в наборе интерфейсов, поддерживаемых в данной конфигурации
bAlternateSetting	Альтернативный номер интерфейса
bNumEndpoints	Число конечных точек для этого интерфейса без учета нулевой конечной точки
bInterfaceClass	Код класса интерфейса
bInterfaceSubClass	Код подкласса интерфейса
bInterfaceProtocol	Код протокола
iInterface	Индекс дескриптора строки, описывающей интерфейс

– **дескриптор конечной точки** (endpoint descriptor) – содержит информацию об одной из конечных точек, доступных при использовании определенного интерфейса. Конечную точку можно представить в виде отдельной функции устройства, например, для съёмного накопителя как правило существует две или более конечных точек: одна для чтения с накопителя, другая для записи. Названия полей и их описания для дескриптора конечной точки показаны в табл. 5.5. Стар-

ший бит поля *bEndpointAddress* определяет направление передачи (0 – от хоста, 1 – к хосту), младшие четыре бита определяют адрес конечной точки, остальные биты должны быть 0. Младшие два бита поля *bmAttributes* определяют тип передачи данных с этой конечной точкой: 00b – control; 01 – isochronous; 10b – bulk; 11 – interrupt.

Таблица 5.5

Поля дескриптора конечной точки

Поле	Описание
bLength	Размер дескриптора в байтах
bDescriptorType	Тип дескриптора (05h)
bEndpointAddress	Код адреса конечной точки
bmAttributes	Атрибуты конечной точки
wMaxPacketSize	Максимальный размер пакета для конечной точки
bInterval	Интервал опроса конечной точки при передаче данных (задается в миллисекундах)

– **дескриптор строки** (string descriptor) – содержит текст в формате Unicode. Строка не ограничивается нулем, а длина строки вычисляется вычитанием 2 из размера дескриптора. Названия полей и их описания для дескриптора строки показаны в табл. 5.6. Дескриптор строки является не обязательным. Если устройство не поддерживает дескрипторы строк, все ссылки (индексы) на такие дескрипторы из дескрипторов устройства, конфигурации или интерфейса должны иметь нулевое значение. Если есть хотя бы один дескриптор строки, то по индексу 0 хранится дескриптор, описывающий идентификаторы языка. Перед получение дескриптора строки с нужным индексом хост должен получить идентификаторы языка, чтобы понимать, какие языки поддерживает устройство.

Таблица 5.6

Поля дескриптора строки

Поле	Описание
bLength	Размер дескриптора в байтах (N+2)
bDescriptorType	Тип дескриптора (03h)
bString	Строка символов (Unicode)

– **сопутствующий дескриптор SS конечной точки** (SuperSpeed endpoint companion descriptor) – содержит информацию о конфигурации конечной точки, работающей на скоростях Gen1 или Gen2. Названия полей и их описания для данного дескриптора показаны в табл. 5.7

Таблица 5.7

Поля сопутствующего дескриптора SS конечной точки

Поле	Описание
bLength	Размер дескриптора в байтах
bDescriptorType	Тип дескриптора (30h)
bMaxBurst	Максимальное число пакетов в очереди, которое может отправить или принять конечная точка
bmAttributes	Атрибуты конечной точки в зависимости от типа передачи данных
wBytesPerInterval	Число передаваемых байт при периодическом опросе конечной точки

В процессе идентификации устройства хост с помощью управляющих посылок запрашивает дескрипторы в следующей последовательности:

- дескриптор устройства;
- дескрипторы конфигураций;
- дескрипторы интерфейсов для каждой конфигурации;
- дескрипторы конечных точек всех интерфейсов;
- сопутствующий дескриптор для SS конечной точки;
- дескрипторы строки.

Как правило, при запросе дескриптора конфигурации, устройство сразу возвращает последовательность из нескольких дескрипторов: конфигураций, интерфейсов и конечных точек.

Для некоторого ряда устройств существуют специальные дескрипторы, которые уточняют параметры работы устройства. Так хабы должны возвращать одноименный дескриптор с информацией о количестве портов и максимальном энергопотреблении, а HID-устройства возвращают серию дескрипторов с информацией о возможностях устройства и способах взаимодействия с ним. Подробную информацию об этих дескрипторах можно найти в [7.1]

2.2. Получение дескрипторов устройств

После идентификации устройства пользовательское приложение может получить USB дескрипторы от драйвера, при этом запрос дескрипторов может проходить в произвольном порядке. Далее представлен исходный код приложения, получающее дескрипторы устройства и конфигурации всех подключенных USB устройств. Программа работает по следующему алгоритму:

1) формируется символьное имя хост-контроллера в виде:

\\.\HCDx,

где x – номер текущего хост-контроллера;

2) по имени хоста определяется его дескриптор (handle);

3) запрашивается символьное имя корневого хаба текущего хост-контроллера;

4) запрашивается дескриптор корневого хаба по символьному имени;

5) определяется количество портов корневого хаба;

6) выполняется перебор всех портов хаба с определением состояния каждого порта;

7) если порт находится в состоянии «подключено» запрашивается дескриптор устройства и дескриптор конфигурации;

8) полученные дескрипторы выводятся в консольное окно.

Поскольку в системе возможно наличие нескольких хост-контроллеров в программе опрашиваются последовательно все, начиная с нулевого, а в случае отрицательного ответа от очередного хоста программа завершает свою работу. При запросе дескриптора конфигурации драйвер в буфере данных возвращает сразу несколько дескрипторов (конфигурации, интерфейса и конечных точек), поэтому приходится перебирать его содержимое для поиска нужного дескриптора. Все возвращаемые USB дескрипторы имеют определённый формат и всегда начинаются с полей размера и типа дескриптора.

```
TITLE USBView32
;32-х разрядное приложение для вывода в консольное окно
; дескрипторов всех подключенных USB устройств.

.386
; Плоская модель памяти и стандартная модель вызова подпрограмм.
.MODEL FLAT, STDCALL

; Директивы компоновщику для подключения библиотек.
INCLUDELIB import32.lib          ; Работа с библиотекой ОС Kernel32.

;--- Внешние WinAPI-функций ----------------------------------------------
EXTRN FreeConsole:        PROC   ; Освободить текущую консоль
EXTRN AllocConsole:       PROC   ; Создать свою консоль
EXTRN GetStdHandle:       PROC   ; Получить дескриптор текущей консоли
EXTRN printf:             PROC   ; Вывести по шаблону
EXTRN CreateFileA:        PROC   ; Получить дескриптор ресурса
EXTRN DeviceIoControl:    PROC   ; Получить информацию об устройстве
EXTRN RtlZeroMemory:      PROC   ; Очистить память
EXTRN ReadConsoleInputA: PROC   ; Получить события консоли
EXTRN GetLastError:       PROC   ; Получить код ошибки
EXTRN CloseHandle:        PROC   ; Закрыть дескриптор
```

```
EXTRN ExitProcess:          PROC   ; Завершить процесс

;--- Константы ----------------------------------------------------------------
GENERIC_WRITE  = 40000000h  ; допускается запись
FILE_SHARE_WRITE      = 2       ; допускается запись другими процессами
OPEN_EXISTING         = 3       ; предписывает открывать устройство,
                                ;если оно существует
STD_INPUT_HANDLE      = -10     ; консольное окно для ввода данных
USB_REQUEST_GET_DESCRIPTOR= 06  ; запрос дескриптора
Key_Event             = 1       ; тип клавиатурного события

; Типы дескрипторов
USB_DEVICE_DESCRIPTOR_TYPE                  = 01h
USB_CONFIGURATION_DESCRIPTOR_TYPE           = 02h

; Управляющие коды операций
IOCTL_USB_GET_ROOT_HUB_NAME                 = 00220408h;
IOCTL_USB_GET_NODE_INFORMATION              = 00220408h;
IOCTL_USB_GET_NODE_CONNECTION_INFORMATION_EX = 00220448h;
IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION = 00220410h;

;--- Структуры данных ---------------------------------------------------------
; Структура символьного имени устройства
TDeviceName STRUC
  Length      dd ?            ; Размер структуры в байтах
  Name        dw 68 dup (?)   ; Символьное имя устройства в Юникоде
TDeviceName ENDS

; Структура дескриптора хаба
THubDescriptor STRUC
  bDescriptorLength       db ?          ; Длина дескриптора
  bDescriptorType         db ?          ; Тип дескриптора
  bNumberOfPorts          db ?          ; Число портов хаба
  wHubCharacteristics     dw ?          ; Атрибуты порта
  bPowerOnToPowerGood     db ?          ; Время стабилизации напряжения
  bHubControlCurrent      db ?          ; Максимальное потребление (мА)
  bRemoveAndPowerMask     db 64 dup (?) ; не используется
THubDescriptor ENDS

; Структура с информацией о хабе
THubInformation STRUC
```

```
  HubDescriptor       THubDescriptor <>
  HubIsBusPowered db ?                ; Способ питания хаба: от шины (1)
                                      ;или от внешнего источника (0)
THubInformation ENDS

; Структура запроса/ответа с информацией о хабе
TNodeInformation STRUC
  NodeType       dd ?                 ; Тип устройства: 0 – хаб
                                      ;1 – комбинир. устройство
  HubInformation THubInformation <>
TNodeInformation ENDS

; Структура дескриптора USB устройства
TDeviceDescriptor STRUC
  bLength             db ?   ; Длина дескриптора
  bDescriptorType     db ?   ; Тип дескриптора
  bcdUSB              dw ?   ; Номер поддерживаемой специф. USB
  bDeviceClass        db ?   ; Код класса USB
  bDeviceSubClass     db ?   ; Код подкласса USB устройства
  bDeviceProtocol     db ?   ; Код протокола USB
  bMaxPacketSize      db ?   ; Макс. размер пакета для нулевой
                             ;конечной точки
  idVendor            dw ?   ; Идентификатор производителя
  idProduct           dw ?   ; Идентификатор продукта
  bcdDevice           dw ?   ; Номер версии устройства
  iManufacturer       db ?   ; Индекс дескриптора строки,
                             ;описывающего производителя
  iProduct            db ?   ; Индекс дескриптора строки,
                             ;описывающего продукт
  iSerialNumber       db ?   ; Индекс дескриптора строки с серийным
                             ;номером устройства
  bNumConfigurations  db ?   ; Число конфигураций у устройства
TDeviceDescriptor ENDS

; Структура с информацией о порте
TNodeConnectionInformation STRUC
  ConnectionIndex           dd ?          ; Индекс порта
  DeviceDescriptor TDeviceDescriptor <>   ; Дескриптор устройства
  CurrentConfigurationValue db ?          ; Номер конфигурации устройства
  Speed                     db ?          ; Скорость работы устройства
                                          ;(0-LS, 1-FS, 2-HS, 3-SS)
```

```
  DeviceIsHub              db ?          ; Индикатор подключенного хаба
  DeviceAddress            dw ?          ; Адрес устройства на шине USB
  NumberOfOpenPipes        db ?          ; Число открытых канал,
                                         ;связанных с портом
  ConnectionStatus         db 4 dup (?)  ; Статус порта (см. ниже)
  PipeList                 db 32 dup (?) ; Массив структур с описание
                                         ;каналов, связанных с портом
TNodeConnectionInformation ENDS
;ConnectionStatus =
;  0 = NoDeviceConnected            - нет подключенных устройств
;  1 = DeviceConnected              - устройство подключено
;  2 = DeviceFailedEnumeration      - ошибка нумерации
;  3 = DeviceGeneralFailure         - фатальная ошибка
;  4 = DeviceCausedOvercurrent      - перегрузка по току
;  5 = DeviceNotEnoughPower         - недостаточное питание
;  6 = DeviceNotEnoughBandwidth     - не хватает пропускной способности
;  7 = DeviceHubNestedTooDeeply     - превышен уровень каскадиров. хабов
;  8 = DeviceInLegacyHub            - не поддерживаемый хаб
;  9 = DeviceEnumerating            - устройство в состоянии «нумерации»
;  10= DeviceReset                  - устройство в состоянии «сброса»

; Структура конфигурационного пакета запроса к устройству
TSetupPacket STRUC
  bmRequest        db ?          ; Тип запроса
  bRequest         db ?          ; Номер запроса
  wValue           dw ?          ; Тип и индекс дескриптора
  wIndex           dw ?          ; Индекс дескриптора
  wLength          dw ?          ; Длина данных при вторичном запросе
TSetupPacket ENDS

; Структура запроса к устройству
UsbDescriptorRequest STRUC
  ConnectionIndex  dd ?          ; Номер объекта (порт, конеч. точка)
  SetupPacket      TSetupPacket <>
  Data             db 2048 dup (?) ; Буфер для возвращаемых данных
UsbDescriptorRequest ENDS

; Структура с информацией о событиях от клавиатуры
KeyEvent STRUC
  KeyDown          dd ?          ; Признак нажатия/отпускания клавиши
  RepeatCount      dw ?          ; Кол-во повторов при удержании клавиши
```

```
  VirtualKeyCode    dw ?          ; Виртуальный код клавиши
  VirtualScanCode   dw ?          ; Скан-код клавиши
  ASCIIChar         dw ?          ; ASCII-код клавиши
  ControlKeyState   dd ?          ; Состояние управляющих клавиш
KeyEvent ENDS

; Структура буфера событий консоли
InputRecord  STRUC
    EventType                dw ?          ; Тип события
                             dw 0          ; для выравнивания поля
    KeyEventRecord       KeyEvent <>
    MouseEventRecord     dd 4 dup (?)
    WindowsBufferSizeRecord dd ?
    MenuEventRecord      dd ?
    FocusEventRecord     dd ?
InputRecord  ENDS

;--- Данные ------------------------------------------------------------------------
.DATA
; Выводимые сообщения
  msgTitle          db 'Получение дескрипторов USB устройств', 13,10,13,10, 0
  msgHostError  db 'Ни одного USB хост-контроллера не найдено', 13, 10, 0
  msgError      db 13,10,'Ошибка: %u', 13, 10, 0
  msgInitHost db 0B3h, 13, 10, 0C3h, 'Хост контроллер №%d', 13, 10, 0
  msgHubInfo db 0B3h, 0C0h, ' Число портов корневого хаба: %d', 13, 10, 0
  msgNoCon  db 0B3h,' ', 0C3h,' К порту %02d устройств не подключено',13,10,0
  msgHubCon db 0B3h, '  ', 0C3h,' К порту %02d подключен хаб', 13, 10, 0
  msgDevCon   db 0B3h,' ',0C3h,' К порту %02d подключено устройство', 13, 10, 0
  msgStatus       db 0B3h,' ',0C3h,' Порт %02d находится в состоянии <%d>', 13,10,0
  msgDevDescr db 0B3h,'  ',0B3h, 13, 10
                  db 0B3h,'  ',0B3h,'   <Дескриптор устройства>', 13, 10
                  db 0B3h,'  ',0B3h,'              bcdUSB = %X.%X', 13, 10
                  db 0B3h,'  ',0B3h,'        bDeviceClass = %u', 13, 10
                  db 0B3h,'  ',0B3h,'     bDeviceSubClass = %u', 13, 10
                  db 0B3h,'  ',0B3h,'     bDeviceProtocol = %u', 13, 10
                  db 0B3h,'  ',0B3h,'      bMaxPacketSize = %u', 13, 10
                  db 0B3h,'  ',0B3h,'           idVendor = %02X%02Xh', 13, 10
                  db 0B3h,'  ',0B3h,'          idProduct = %02X%02Xh', 13, 10
                  db 0B3h,'  ',0B3h,'          bcdDevice = %02X%02Xh', 13,10
                  db 0B3h,'  ',0B3h,'      iManufacturer = %u', 13, 10
                  db 0B3h,'  ',0B3h,'           iProduct = %u', 13, 10
```

```
                db 0B3h,'  ',0B3h,'       iSerialNumber = %u', 13, 10
                db 0B3h,'  ',0B3h,'  bNumConfigurations = %u', 13, 10, 0
msgConfDescr db 0B3h,'  ',0B3h, 13, 10
                db 0B3h,'  ',0B3h,'    <Дескриптор конфигурации>',13,10
                db 0B3h,'  ',0B3h,'          wTotalLength = %u', 13, 10
                db 0B3h,'  ',0B3h,'       bNumInterfaces = %u', 13, 10
                db 0B3h,'  ',0B3h,'  bConfigurationValue = %u', 13, 10
                db 0B3h,'  ',0B3h,'        iConfiguration = %u', 13, 10
                db 0B3h,'  ',0B3h,'          bmAttributes = %02Xh', 13, 10
                db 0B3h,'  ',0B3h,'             MaxPower = %u мА', 13, 10
                db 0B3h,'  ',0B3h, 13, 10, 0
msgExit      db 13,10,'Для выхода нажмите любую клавишу...', 0

; Переменные
  NameHost    db '\\.\HCD'          ; Имя хоста в Windows
  NumHost     dd ?                  ; Номер хоста
  iHost       dd ?                  ; Индекс хоста
  NameRootHub db '\\.\'             ; Заготовка под имя корневого хаба
  SRootHub    db 136 dup (?)        ; Строка с именем хаба
  hHost       dd ?                  ; Дескриптор хост-контроллера
  hRootHub    dd ?                  ; Дескриптор корневого хаба
  hPort       dd ?                  ; Дескриптор порта
  iPort       dd ?                  ; Индекс текущего порта хаба
  hIn         dd ?                  ; Дескриптор окна ввода данных
  NumEvent    dd ?                  ; Число событий
  bReturned   dd ?                  ; Число байт, возвращённых функцией

; Указатели на структуры
  CBuffer           InputRecord <>                ; буфер консоли
  DeviceName        TDeviceName <>                ; имя устройства
  NodeInformation   TNodeInformation <>           ; информация о хабе
  NodeConnInfo   TNodeConnectionInformation <>    ; информация о порте
  DescriptorRequest  UsbDescriptorRequest <>      ; запрос дескриптора

.CODE
;--- Получение информации о порте хаба -------------------------------------
;hHub – дескриптор корневого хаба
;Port   – индекс порта хаба
;Возвращает EAX = 1 – успешное завершение, 0 – ошибка
ShowHubPortDetail PROC
        ARG   hHub: DWORD, Port: DWORD
```

```
; Получение состояния порта хаба
        mov     eax, Port
        mov     NodeConnInfo.ConnectionIndex, eax
        call    DeviceIoControl, hHub,
IOCTL_USB_GET_NODE_CONNECTION_INFORMATION_EX,
offset NodeConnInfo, size NodeConnInfo, offset NodeConnInfo,
size NodeConnInfo, offset bReturned, 0
        test    eax, eax    ; Проверка успешного выполнения команды
        jz      SHPDExit ; В случае ошибки выход из процедуры (eax=0)

; Проверка статуса порта
        cmp     NodeConnInfo.ConnectionStatus[3], 0
        jne     ConDev
        call    printf, offset msgNoCon, Port
        add     esp, 4*2
        jmp     NoError
ConDev:
        cmp     NodeConnInfo.ConnectionStatus[3], 1
        jne     NextStatus
        cmp     NodeConnInfo.DeviceIsHub, 0
        jz      NoHub
        call    printf, offset msgHubCon, Port
        add     esp, 4*2
        call    ShowDeviceDetail, hHub, Port
        jmp     NoError
NoHub:
        call    printf, offset msgDevCon, Port
        add     esp, 4*2
        call    ShowDeviceDetail, hHub, Port
        jmp     NoError
NextStatus:
   call  printf, offset msgStatus, Port, dword ptr NodeConnInfo.ConnectionStatus[3]
        add     esp, 4*3

NoError:
        mov     eax, 1
SHPDExit:   ret
ShowHubPortDetail ENDP

;--- Получение USB дескрипторов ----------------------------------------------
```

```
;hHub – дескриптор корневого хаба
;Port   – индекс порта хаба
;Возвращает EAX = 1 – успешное завершение, 0 – ошибка
ShowDeviceDetail PROC
        ARG   hHub: DWORD, Port: DWORD

; Получение дескриптора устройства
        mov     eax, Port
        mov     DescriptorRequest.ConnectionIndex, eax
        mov     DescriptorRequest.SetupPacket.bmRequest, 80h
        mov     DescriptorRequest.SetupPacket.bRequest,
USB_REQUEST_GET_DESCRIPTOR
        mov     DescriptorRequest.SetupPacket.wValue,
USB_DEVICE_DESCRIPTOR_TYPE
        shl     DescriptorRequest.SetupPacket.wValue, 8
        mov     DescriptorRequest.SetupPacket.wLength, 100h

        call    DeviceIoControl, hHub,
IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION,
offset DescriptorRequest, size DescriptorRequest, offset DescriptorRequest,
size DescriptorRequest, offset bReturned, 0
        test    eax, eax
        jz      SDDExit

; Вывод дескриптора устройства
        call    DisplayDescriptorInfo, offset DescriptorRequest.Data

; Очистка буфера
        call    RtlZeroMemory, offset DescriptorRequest.Data,
dword ptr size DescriptorRequest.Data

; Получение дескрипторов конфигураций
        mov     eax, Port
        mov     DescriptorRequest.ConnectionIndex, eax
        mov     DescriptorRequest.SetupPacket.bmRequest, 80h
        mov     DescriptorRequest.SetupPacket.bRequest,
USB_REQUEST_GET_DESCRIPTOR
        mov     DescriptorRequest.SetupPacket.wValue,
USB_CONFIGURATION_DESCRIPTOR_TYPE
        shl     DescriptorRequest.SetupPacket.wValue, 8
        mov     DescriptorRequest.SetupPacket.wLength, 100h
```

```
        call    DeviceIoControl, hHub,
IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION,
offset DescriptorRequest, size DescriptorRequest, offset DescriptorRequest,
size DescriptorRequest, offset bReturned, 0
        test    eax, eax
        jz      SDDExit

; Вывод дескриптора
        call    DisplayDescriptorInfo, offset DescriptorRequest.Data

; Очистка буфера
        call    RtlZeroMemory, offset DescriptorRequest.Data,
dword ptr size DescriptorRequest.Data
        mov     eax, 1
SDDExit:    ret
ShowDeviceDetail ENDP

;--- Вывод дескрипторов в консоль ----------------------------------------------
;Buffer – адрес массива дескрипторов
;edi    – адрес текущего байта в буфере
;esi    – длина дескриптора
DisplayDescriptorInfo PROC
        ARG   Buffer: DWORD
        uses    eax, ebx, edi, esi

; Проходим по массиву дескрипторов
; Первый байт – размер дескриптора в байтах
; Второй байт – тип дескриптора
        mov     edi, Buffer
Cycle:
        movzx esi, byte ptr [edi+0]     ; Длина дескриптора
        cmp     esi, 0                  ; Нет больше дескрипторов?
        je      DDIExit
        cmp     byte ptr [edi+1], 0     ; Нет данных?
        je      DDIExit

        cmp     byte ptr [edi+1], 1     ; Дескриптор устройства?
        jne     DesCon
; Занесение в стек значений в обратном порядке для передачи
;параметров printf
```

```
        xor     eax, eax
        mov     al, [edi+17]
        push    eax
        mov     al, [edi+16]
        push    eax
        mov     al, [edi+15]
        push    eax
        mov     al, [edi+14]
        push    eax
        mov     al, [edi+12]
        push    eax
        mov     al, [edi+13]
        push    eax
        mov     al, [edi+10]
        push    eax
        mov     al, [edi+11]
        push    eax
        mov     al, [edi+8]
        push    eax
        mov     al, [edi+9]
        push    eax
        mov     al, [edi+7]
        push    eax
        mov     al, [edi+6]
        push    eax
        mov     al, [edi+5]
        push    eax
        mov     al, [edi+4]
        push    eax
        mov     al, [edi+2]
        push    eax
        mov     al, [edi+3]
        push    eax
        call    printf, offset msgDevDescr
        add     esp, 4*17
        jmp     NextDescr

DesCon:
        cmp     byte ptr [edi+1], 2  ; Дескриптор конфигурации?
        jne     NextDescr
        xor     eax, eax
```

```
        mov     al, [edi+8]
        mov     bl, 2
        mul     bl                  ; Преобразование значения в мА
        push    eax
        xor     eax, eax
        mov     al, [edi+7]
        push    eax
        mov     al, [edi+6]
        push    eax
        mov     al, [edi+5]
        push    eax
        mov     al, [edi+4]
        push    eax
        mov     ax, [edi+2]
        push    eax
        call    printf, offset msgConfDescr
        add     esp, 4*7

NextDescr:
        add     edi, esi
        jmp     Cycle
DDIExit:    ret
DisplayDescriptorInfo ENDP

;--- Основной код ----------------------------------------------------------------
Start:
        call    FreeConsole
        call    AllocConsole
        call    GetStdHandle, STD_INPUT_HANDLE
        mov     hIn, eax
        call    printf, offset msgTitle
        add     esp, 4*1            ; Корректировка стека после printf

; Цикл по всем хостам
        mov     iHost, 0
mNextHost:
; Формирование имени хоста
        mov     eax, iHost
        add     eax, '0'            ; Перевод номера диска в символ '0', '1' и т.д.
        mov     NumHost, eax        ; Не более 9 хостов в системе
```

```
; Получение дескриптора текущего хоста
        call    CreateFileA,    offset    NameHost,    GENERIC_WRITE,
FILE_SHARE_WRITE, 0, OPEN_EXISTING, 0, 0
        cmp     eax, -1                 ; В случае ошибки вывод сообщения
        je      mErrorHost
        mov     hHost, eax              ; Сохранение дескриптора в hHost

        call    printf, offset msgInitHost, iHost
        add     esp, 4*2

; Получение PnP-имени корневого хаба
        call    DeviceIoControl, hHost,
IOCTL_USB_GET_ROOT_HUB_NAME, 0, 0, offset DeviceName,
size DeviceName, offset bReturned, 0
        test    eax, eax
        jz      mError
        mov     ebx, DeviceName.Length
        cmp     ebx, bReturned          ; Проверка на полностью
        jb      mError                  ; полученную структуру с именем

; Формирование полного имени корневого хаба из PnP-имени
        mov     ecx, 136                ; Максимальная длина имени хаба
        cld                             ; Движение по строке вперёд
        push    ds
        pop     es
        lea     esi, DeviceName.Name    ; Каждый символ занимает 2 байта
        lea     edi, SRootHub           ; Каждый символ занимает 1 байт
mLoop:
        movsb
        inc     esi
        loop    mLoop

; Получение дескриптора текущего корневого хаба
        call    CreateFileA,    offset    NameRootHub,    GENERIC_WRITE,
FILE_SHARE_WRITE, 0, OPEN_EXISTING, 0, 0
        cmp     eax, -1
        je      mError
        mov     hRootHub, eax       ; Сохранение дескриптора в hRootHub

; Получение информации о корневом хабе
        mov     NodeInformation.NodeType, 0
```

```
        call    DeviceIoControl, hRootHub,
IOCTL_USB_GET_NODE_INFORMATION, offset NodeInformation,
size NodeInformation, offset NodeInformation, size NodeInformation,
offset bReturned, 0
        test    eax, eax
        jz      mError

        xor     ebx, ebx            ; ebx – число портов хаба
        movzx ebx,
NodeInformation.HubInformation.HubDescriptor.bNumberOfPorts
        call    printf, offset msgHubInfo, ebx
        add     esp, 4*2

; Перебор всех портов хаба
        mov     iPort, 1
mNextPort:
        call    ShowHubPortDetail, hRootHub, iPort;
        test    eax, eax
        jz      mError

        inc     iPort
        cmp     ebx, iPort
        jnb     mNextPort

        call    CloseHandle, hRootHub
        call    CloseHandle, hHost
        inc     iHost
        jmp     mNextHost

mErrorHost:                         ; Вывод последнего сообщения
        call    printf, offset msgExit
        add     esp, 4*1
        jmp     mWait

mError:                             ; Обработка ошибок
        call    GetLastError
        call    printf, offset msgError, eax
        add     esp, 4*2
        cmp     hRootHub, 0
        je      mWait
        call    CloseHandle, hRootHub
```

```
        call    CloseHandle, hHost

mWait:                          ; Ожидание нажатия клавиши
        call    ReadConsoleInputA, hIn, offset CBuffer, 1, offset NumEvent
        cmp     CBuffer.EventType, Key_Event
        jne     mWait           ; Событие от клавиатуры?

mExit:                          ; Выход из программы
        call    CloseHandle, hIn
        call    FreeConsole
        call    ExitProcess, 0
END Start
```

2.3. Утилиты для идентификации устройств и мониторинга USB интерфейса

Для получения информации обо всех установленных в компьютере USB контроллерах и подключенных к ним устройствах можно воспользоваться бесплатной программой **USB Device Tree Viewer** (USBTreeView) от Uwe Sieber [7.2], которая реализована на базе исходников программы **USB Device Viewer** от компании Microsoft Corporation [7.3]. **USBTreeView** отображает древовидную структуру подключенных к компьютеру USB устройств (в левой части окна) и содержимое их дескрипторов (справа), а также дополнительную информацию полученную из реестра и других источников.

Для мониторинга и анализа данных передаваемых по USB интерфейсу можно воспользоваться связкой двух бесплатных программ: **USBPcap** и **Wireshark**. **USBPcap** [7.4] позволяет выполнять захват пакетов передаваемых по USB, а **Wireshark** [7.5] отображать в наглядном виде полученные данные.

3. Подготовка к работе

3.1. Изучить методические указания и рекомендованную литературу.

3.2. Подготовить ответы на контрольные вопросы.

4. Задание на выполнение работы

4.1. Подключить к компьютеру устройство с USB-интерфейсом (например, манипулятор «мышь», съёмный флэш-диск и т.п.).

4.2. Запустить программу **USB Device Tree Viewer**.

4.3. Для подключенного устройства просмотреть и проанализировать его дескрипторы и всю выводимую программой информацию. Заполнить табл. 5.8.

4.4. Создать и отладить исполняемый модуль программы **USBView32**, исходный код которой приведён в разделе 2.2. Проверить правильность выводимой информации с программой **USB Device Tree Viewer.**

4.5. Отредактировать исходный модуль программы **USBView32** таким образом, чтобы выполнить задание в соответствии с вариантом из табл. 5.9.

Вариант задания соответствует целой части суммы деления последней цифры зачетной книжки на два с единицей. Пример: номер зачетной книжки 1234567, тогда номер варианта находится из выражения N=7/2+1, в этом случае номер варианта равен 4.

Таблица 5.8

Сведения о USB устройстве

Характеристика	Значение
Идентификатор производителя	
Идентификатор продукта	
Серийный номе (если есть)	
Число конфигураций	
Число интерфейсов в конфигурации	
Питается от шины USB (да/нет)	
Поддерживает режим пробуждения (да/нет)?	
Максимальное токопотребление, мА	
Количество конечных точек у устройства	
Код класса интерфейса	
Тип передачи данных первой конечной точки устройства	
Тип передачи данных второй конечной точки устройства (если есть)	
Номер поддерживаемой спецификации USB	
Скорость работы устройства (LS, FS, HS, SS)	
Поддерживает скорость SS Plus (SS Gen 2)?	

5. Требования к отчёту

Отчёт должен содержать:

– титульный лист с указанием названия ВУЗа, кафедры, номера и темы лабораторной работы, а также фамилии И.О. студента, подготовившего отчёт;

– цель работы;

– информацию о USB устройстве, занесённую в табл. 5.8.;

– листинг отредактированной программы **USBView32** с результатами её работы.

6. Контрольные вопросы

6.1. Логическая и физическая архитектура USB.

6.2. Из каких элементов состоит шина USB?

6.3. Какие скорости передачи данных определяет стандарт USB?

6.4. Что такое USB дескриптор?

Таблица 5.9

Варианты заданий

№ варианта	Задание
1	Получить и вывести на экран все основные дескрипторы (стандартный, конфигурации, интерфейса, конечной точки) вашего USB устройства
2	Определить и вывести на экран тип передачи данных для всех конечных точек всех подключенных устройств
3	Определить и вывести на экран коды класса всех подключенных устройств и общее число устройств соответствующего класса
4	Определить и вывести на экран общее число хост-контроллеров, общее число портов всех корневых хабов, общее число подключенных устройств
5	Определить и вывести на экран по каждому подключенному устройству его адрес, идентификатор производителя и продукта, а также скорость на которой работает устройство

6.5. Какую информацию содержит стандартный дескриптор USB устройства?
6.6. С помощью чего однозначно определяется модель USB устройства?
6.7. Что называют конечной точкой USB устройства?
6.8. Какие существуют типы USB дескрипторов и для чего они предназначены?
6.9. В какой последовательности USB хост запрашивает дескрипторы у устройства?
6.10. Как определить скорость, на которой работает USB устройство?
6.11. В каком дескрипторе содержится число конечных точек USB устройства?

7. Рекомендуемая литература

7.1. Агуров, П.В. Практика программирования USB [Текст] / П.В. Агуров. – СПб.: БХВ-Петербург, 2006. с. 52…72.
7.2. USB Device Tree Viewer. [Электронный ресурс] / Uwe Sieber. – Режим доступа: http://www.uwe-sieber.de/usbtreeview_e.html, свободный – 08.11.2017.
7.3. USBView | Microsoft Docs [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/usbview, свободный – 08.11.2017.
7.4. USBPcap [Электронный ресурс] – Режим доступа: http://desowin.org/usbpcap/index.html, свободный – 08.11.2017.
7.5. Wireshark Go Deep [Электронный ресурс] – Режим доступа: https://www.wireshark.org, свободный – 08.11.2017.