МИНОБРНАУКИ РОССИИ
Государственное образовательное учреждение высшего профессионального образования
«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ»
(СПбГЭТУ)

А.Ф.ГУБКИН

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ
по дисциплине "Операционные системы"

Санкт-Петербург
2011

ВВЕДЕНИЕ

Цикл лабораторных работ по дисциплине "Операционные системы" предназначен для получения студентами практических навыков работы с управляющей программой операционной системы (ОС) на уровне программного интерфейса. В лабораторных работах исследуются структуры загрузочных модулей, интерфейс прикладных программ с управляющей программой, управление основной памятью, резидентные обработчики прерываний, загрузка приложений разных форматов для выполнения.

Лабораторные работы выполняются в широко доступной среде Windows XP, не требующей привилегированного доступа пользователей и специальных инструментов.

Работы предполагают знания студентами языка ассемблер Intel и умения работать с системой программирования Microsoft assembler (MASM) или Turbo assembler (TASM), использования компоновщика Link (TLINK) и отладчика TD.

Выполнение лабораторных работ состоит в исследовании механизмов управляющей программы ОС с помощью написанных на ассемблере утилит. В некоторых работах требуется создать несколько вариантов утилиты, с целью исследования различных возможностей, предоставляемых функциями управляющей программы.

По результатам выполнения лабораторной работы представляются исходные тексты утилит, загрузочные модули, отчет в электронном виде в формате MS Word 2003, содержащий постановку задачи и материалы в виде выходной информации утилит и скриншотов, обосновывающие ответы на поставленные вопросы.

ОБЩИЕ СВЕДЕНИЯ

Общие сведения содержат информацию, используемую во всех лабораторных работах и общие требования, которые должны выполняться во всех работах.

Реализация программного интерфейса

Программный интерфейс реализуется посредством функций, доступ к которым осуществляется по программному (синхронному) прерыванию с вектором 21h. Программные прерывания выполняются синхронно в темпе выполнения программы. В результате прерывания управление передается ядру операционной системы и по коду функции, заданной в регистре как параметр, обработчик прерываний выбирает соответствующую функцию и передает ей управление.

Обращение к функциям управляющей программы

Обращение к функциям реализуется следующим образом. В регистры в зависимости от функции заносятся различные параметры. Однако, в регистр AH всегда заносится код вызываемой функции. Затем выполняется прерывание с вектором 21h. Пример текста на ассемблере:

```
mov     DX,offset STRING ;занесение параметра функции
mov     AH,09h           ;занесение кода функции
int 21h                  ;прерывание – обращение к
                         ;функции
```

Обработка завершения функции управляющей программы

После обращения к функции необходимо обработать завершение выполнения функции. Для этого следует проверить C-бит в регистре флагов. Если C-бит имеет значение 0, то функция выполнена успешно. Если C-бит имеет значение 1, то в регистре AX содержится код завершения, который определяет ошибку. В этом случае программа должна завершить работу и вывести содержимое регистра AX в шестнадцатеричном и десятичном виде, а также диагностическое сообщение.

Ниже приведены некоторые коды завершения (RC) и диагностические сообщения. Если вызываемая функция не была выполнена, то устанавливается флаг переноса CF=1 и в AX заносится код ошибки:

- 1 - если номер функции неверен;
- 2 - если файл не найден;
- 5 - при ошибке диска;
- 8 - при недостаточном объеме памяти;
- 10 - при неправильной строке среды;
- 11 - если не верен формат.

Требуемая структура исходного текста модуля типа .COM

Шаблон ассемблерного текста с функциями управляющей программы и процедурами перевода двоичных кодов в символы шестнадцатеричных чисел и десятичное число приводится ниже:

```
; Шаблон текста программы на ассемблере для модуля типа .COM
TESTPC     SEGMENT
           ASSUME  CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
           ORG     100H
START:     JMP     BEGIN
; ДАННЫЕ
STRING     db      'Значение регистра AX=     ',0DH,0AH,'$'
;ПРОЦЕДУРЫ
;-----------------------------------------------
TETR_TO_HEX   PROC  near
```

```
            and     AL,0Fh
            cmp     AL,09
            jbe     NEXT
            add     AL,07
NEXT:       add     AL,30h
            ret
TETR_TO_HEX   ENDP
;--------------------------------
BYTE_TO_HEX   PROC  near
; байт в AL переводится в два символа шестн. числа в AX
            push    CX
            mov     AH,AL
            call    TETR_TO_HEX
            xchg    AL,AH
            mov     CL,4
            shr     AL,CL
            call    TETR_TO_HEX ;в AL старшая цифра
            pop     CX          ;в AH младшая
            ret
BYTE_TO_HEX  ENDP
;--------------------------------
WRD_TO_HEX   PROC  near
;перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
            push    BX
            mov     BH,AH
            call    BYTE_TO_HEX
            mov     [DI],AH
            dec     DI
            mov     [DI],AL
            dec     DI
            mov     AL,BH
            call    BYTE_TO_HEX
            mov     [DI],AH
            dec     DI
            mov     [DI],AL
            pop     BX
            ret
WRD_TO_HEX ENDP
;------------------------------------------------
BYTE_TO_DEC   PROC  near
; перевод в 10с/с, SI - адрес поля младшей цифры
            push    CX
            push    DX
            xor     AH,AH
            xor     DX,DX
            mov     CX,10
loop_bd:    div     CX
            or      DL,30h
            mov     [SI],DL
            dec     SI
            xor     DX,DX
            cmp     AX,10
            jae     loop_bd
            cmp     AL,00h
            je      end_l
            or      AL,30h
            mov     [SI],AL
end_l:      pop     DX
            pop     CX
            ret
BYTE_TO_DEC    ENDP
;--------------------------------
; КОД
```

```
BEGIN:
        . . . . . .
; Вывод строки текста из поля STRING
          mov     DX,offset STRING
          mov     AH,09h
          int     21h

        . . . . . .
; Выход в DOS
          xor     AL,AL
          mov     AH,4Ch
          int     21H
TESTPC    ENDS
          END     START     ;конец модуля, START – точка входа
```

Использование функции 4Ah

Использование функции 4Ah позволяет освободить неиспользуемую программой память, поскольку при загрузке модуля вся память выделяется этому модулю, в соответствии со стратегией однопрограммных ОС.

Для этого следует использовать функции 4Ah прерывания 21h. Пример обращения к этой функции:

```
MOV     AH,4AH
MOV     BX,mem-size  ;размер памяти программы в параграфах
INT     21H
```

В регистр BX заносится размер памяти в параграфах, который необходимо оставить программе.

Если занести заведомо больший размер памяти, чем может предоставить ОС, то в регистре BX возвращается размер доступной памяти в параграфах. В этом случае следует использовать следующее обращение:

```
MOV         AH,4AH
MOV         BX,0FFFFH         ; заведомо большая память
INT         21H
```

Если вызываемая функция не была выполнена, то устанавливается флаг переноса CF=1 и в AX заносится код ошибки.

Требования к оформлению отчета

Отчет выполняется в формате MS Word2003 и представляется в электронном виде единым файлом. Отчет должен содержать:

1) Титульный лист с названием лабораторной работы, фамилией, именем и отчеством студента, выполнившего работу, номером группы и названием факультета, фамилия, И.О. преподавателя, датой выполнения работы.
2) Первый раздел «Постановка задачи» должен содержать формулировку цели работы, сведения о функциях и структурах данных управляющей программы, используемых в работе, последовательность действий, выполняемых утилитой.
3) Описание результатов исследования проблем, поставленных в лабораторной работе. По каждой работе в Методических Указаниях приводятся вопросы, на которые необходимо найти ответы и подтвердить их соответствующей информацией. В отчете приводятся скриншоты, подтверждающие выполнения шагов лабораторной работы и ответы на вопросы.
4) Заключение.

По результатам выполнения лабораторной работы представляются;

1) исходные тексты программ,
2) загрузочные модули,

3) отчет в электронном виде по установленной форме, содержащий постановку задачи и материалы в виде выходной информации утилит и скриншотов, обосновывающие ответы на поставленные вопросы.

Требования к защите лабораторной работы

Отчет по лабораторной работе, исходные тексты разработанных утилит и загрузочные модули присылаются для проверки по E-mail на адрес:

gubkin_alexandr@mail.ru

После возможного диалога по поводу присланных материалов после достижения согласия ставится зачет, который подтверждается письмом.

Получение зачета по всем лабораторным работам позволяет получить общий зачет.

ЛАБОРАТОРНАЯ РАБОТА № 1
«Исследование структур загрузочных модулей»

Цель работы: Исследование различий в структурах исходных текстов модулей типов **.COM** и **.EXE**, структур файлов загрузочных модулей и способов их загрузки в основную память.

Порядок выполнения работы

«Истина познается в сравнении», как говорили древние. К счастью, у нас есть возможность исследовать в одной системе два различных формата загрузочных модулей, сравнить их и лучше понять как система программирования и управляющая программа обращаются с ними. Система программирования включает компилятор с языка ассемблер (часто называется, просто, ассемблер), который изготавливает объектные модули. Компоновщик (Linker) по совокупности объектных модулей, изготавливает загрузочный модуль, а также, функция ядра – загрузчик, которая помещает программу в основную память и запускает на выполнение. Все эти компоненты согласованно работают для изготовления и выполнения загрузочных модулей разного типа. Для выполнения лабораторной работы сначала нужно изготовить загрузочные модули.

Шаг 1. Напишите текст исходного **.COM** модуля, который определяет тип PC и версию системы. Это довольно простая задача и для тех, кто уже имеет опыт программирования на ассемблере, это будет небольшой разминкой. Для тех, кто раньше не сталкивался с программированием на ассемблере, это неплохая задача для первого опыта.

За основу возьмите шаблон, приведенный в разделе «Основные сведения». Необходимые сведения о том, как извлечь требуемую информацию, представлены в следующем разделе.

Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип PC и выводить строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводиться в символьную строку, содержащую запись шестнадцатеричного числа и выводиться на экран в виде соответствующего сообщения.

Затем определяется версия системы. Ассемблерная программа должна по значениям регистров AL и AH формировать текстовую строку в формате xx.yy, где xx - номер основной версии, а yy - номер модификации в десятичной системе счисления, формировать строки с серийным номером OEM и серийным номером пользователя. Полученные строки выводятся на экран.

Отладьте полученный исходный модуль.

Результатом выполнения этого шага будет «хороший» **.COM** модуль, а также необходимо построить «плохой» **.EXE**, полученный из исходного текста для **.COM** модуля.

Шаг 2. Напишите текст исходного **.EXE** модуля, который выполняет те же функции, что и модуль в Шаге 1 и постройте и отладьте его. Таким образом, будет получен «хороший» **.EXE.**

Шаг 3. Сравните исходные тексты для **.COM** и **.EXE** модулей. Ответьте на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

Шаг 4. Запустите FAR и откройте (F3/F4) файл загрузочного модуля **.COM** и файл «плохого» **.EXE** в шестнадцатеричном виде. Затем откройте (F3/F4) файл загрузочного модуля «хорошего» **.EXE** и сравните его с предыдущими файлами. Ответьте на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

Шаг 5. Откройте отладчик **TD.EXE** и загрузите **.COM.** Ответьте на контрольные вопросы «Загрузка COM модуля в основную память». Представьте в отчете план загрузки модуля **.COM** в основную память.

Шаг 6. Откройте отладчик **TD.EXE** и загрузите «хороший» **.EXE.** Ответьте на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

Шаг 7. Оформление отчета в соответствии с требованиями. В отчете необходимо привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей – в отладчике.

Необходимые сведения для составления программы

Тип IBM PC хранится в байте по адресу 0F000:0FFFEh, в предпоследнем байте ROM BIOS. Соответствие кода и типа в таблице:

PC	**FF**
PC/XT	**FE,FB**
AT	**FC**
PS2 модель 30	**FA**
PS2 модель 50 или 60	**FC**
PS2 модель 80	**F8**
PCjr	**FD**
PC Convertible	**F9**

Для определения версии MS DOS следует воспользоваться функцией 30H прерывания 21H. Входным параметром является номер функции в AH:

```
MOV AH,30h
INT 21h
```

Выходными параметрами являются:

AL - номер основной версии. Если 0, то < 2.0
AH - номер модификации
BH - серийный номер OEM (Original Equipment Manufacturer)
BL:CX - 24-битовый серийный номер пользователя.

Контрольные вопросы по лабораторной работе №1

Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-программа?
2) EXE-программа?
3) Какие директивы должны обязательно быть в тексте COM-программы?
4) Все ли форматы команд можно использовать в COM-программе?

Отличия форматов файлов COM и EXE модулей

1) Какова структура файла COM? С какого адреса располагается код?
2) Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с адреса 0?
3) Какова структура файла «хорошего» EXE? Чем он отличается от файла «плохого» EXE?

Загрузка COM модуля в основную память

1) Какой формат загрузки модуля COM? С какого адреса располагается код?
2) Что располагается с адреса 0?
3) Какие значения имеют сегментные регистры? На какие области памяти они указывают?
4) Как определяется стек? Какую область памяти он занимает? Какие адреса?

Загрузка «хорошего» EXE модуля в основную память

1) Как загружается «хороший» EXE? Какие значения имеют сегментные регистры?
2) На что указывают регистры DS и ES?
3) Как определяется стек?
4) Как определяется точка входа?

ЛАБОРАТОРНАЯ РАБОТА № 2
«Исследование интерфейсов программных модулей»

Цель работы: Исследование интерфейса управляющей программы и загрузочных модулей. Этот интерфейс состоит в передаче запускаемой программе управляющего блока, содержащего адреса и системные данные. Так загрузчик строит префикс сегмента программы (PSP) и помещает его адрес в сегментный регистр. Исследование префикса сегмента программы (PSP) и среды, передаваемой программе.

Порядок выполнения работы

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа **.COM,** который выбирает и распечатывает следующую информацию:

1) Сегментный адрес недоступной памяти, взятый из PSP, в шестнадцатеричном виде.
2) Сегментный адрес среды, передаваемой программе, в шестнадцатеричном виде.
3) Хвост командной строки в символьном виде.
4) Содержимое области среды в символьном виде.
5) Путь загружаемого модуля.

Сохраните результаты, полученные программой, и включите их в отчет.

Шаг 2. Оформление отчета в соответствии с требованиями. В отчет включите скриншот с запуском программы и результатами.

Необходимые сведения для составления программы

При начальной загрузке программы формируется PSP, который размещается в начале первого сегмента программы. PSP занимает 256 байт и располагается с адреса, кратного границе сегмента. При загрузке модулей типа **.COM** все сегментные регистры указывают на адрес PSP. При загрузке модуля типа **.EXE** сегментные регистры DS и ES указывают на PSP. Именно по этой причине значения этих регистров в модуле **.EXE** следует переопределять.

Формат PSP:

Смещение	Длина поля(байт)	Содержимое поля
0	2	int 20h
2	2	Сегментный адрес первого байта недоступной памяти. Программа не должна модифицировать содержимое памяти за этим адресом.
4	6	Зарезервировано
0Ah (10)	4	Вектор прерывания 22h (IP,CS)
0Eh (14)	4	Вектор прерывания 23h (IP,CS)
12h (18)	4	Вектор прерывания 24h (IP,CS)
2Ch (44)	2	Сегментный адрес среды, передаваемой программе.
5Ch		Область форматируется как стандартный неоткрытый блок управления файлом (FCB)
6Ch		Область форматируется как стандартный неоткрытый блок управления файлом (FCB). Перекрывается, если FCB с адреса 5Ch открыт.
80h	1	Число символов в хвосте командной строки.
81h		Хвост командной строки - последовательность символов после имени вызываемого модуля.

Область среды содержит последовательность символьных строк вида:

```
имя=параметр
```

Каждая строка завершается байтом нулей.

В первой строке указывается имя COMSPEC, которая определяет используемый командный процессор и путь к COMMAND.COM. Следующие строки содержат информацию, задаваемую командами PATH, PROMPT, SET.

Среда заканчивается также байтом нулей. Таким образом, два нулевых байта являются признаком конца переменных среды. Затем идут два байта, содержащих 00h, 01h, после которых располагается маршрут загруженной программы. Маршрут также заканчивается байтом 00h.

Контрольные вопросы по лабораторной работе №2

Сегментный адрес недоступной памяти

1) На какую область памяти указывает адрес недоступной памяти?
2) Где расположен этот адрес по отношению области памяти, отведенной программе?
3) Можно ли в эту область памяти писать?

Среда передаваемая программе

1) Что такое среда?
2) Когда создается среда? Перед запуском приложения или в другое время?
3) Откуда берется информация, записываемая в среду?

ЛАБОРАТОРНАЯ РАБОТА № 3
«Исследование организации управления основной памятью»

Цель работы: Для исследования организации управления памятью необходимо ориентироваться на тип основной памяти, реализованный в компьютере и способ организации, принятый в ОС. В лабораторной работе рассматривается нестраничная память и способ управления динамическими разделами. Для реализации управления памятью в этом случае строится список занятых и свободных участков памяти. Функции ядра, обеспечивающие управление основной памятью, просматривают и преобразуют этот список.

В лабораторной работе исследуются структуры данных и работа функций управления памятью ядра операционной системы.

Порядок выполнения работы

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа **.COM,** который выбирает и распечатывает следующую информацию:

1) Количество доступной памяти.
2) Размер расширенной памяти.
3) Выводит цепочку блоков управления памятью.

Адреса при выводе представляются шестнадцатеричными числами. Объем памяти функциями управления памятью выводится в параграфах. Необходимо преобразовать его в байты и выводить в виде десятичных чисел. Последние восемь байт MCB выводятся как символы, не следует преобразовывать их в шестнадцатеричные числа.

Запустите программу и внимательно оцените результаты. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

Шаг 2. Измените программу таким образом, чтобы она освобождала память, которую она не занимает. Для этого используйте функцию 4Ah прерывания 21h (пример в разделе «Использование функции 4AH»). Повторите эксперимент, запустив модифицированную программу. Сравните выходные данные с результатами, полученными на предыдущем шаге. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

Шаг 3. Измените программу еще раз таким образом, чтобы после освобождения памяти, программа запрашивала 64Кб памяти функцией 48H прерывания 21H. Повторите эксперимент, запустив модифицированную программу. Сравните выходные данные с результатами, полученными на предыдущих шагах. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

Шаг 4. Измените первоначальный вариант программы, запросив 64Кб памяти функцией 48H прерывания 21H до освобождения памяти. Обязательно обрабатывайте завершение функций ядра, проверяя флаг CF. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

Шаг 5. Оцените результаты, полученные на предыдущих шагах. Ответьте на контрольные вопросы и оформите отчет.

Необходимые сведения для составления программы

Учет занятой и свободной памяти ведется при помощи списка блоков управления памятью MCB (Memory Control Block). MCB занимает 16 байт (параграф) и располагается всегда с адреса кратного 16 (адрес сегмента ОП) и находится в адресном пространстве непосредственно перед тем участком памяти, которым он управляет.

MCB имеет следующую структуру:

Смещение	Длина поля (байт)	Содержимое поля
00h	1	тип MCB: 5Ah, если последний в списке, 4Dh, если не последний
01h	2	Сегментный адрес PSP владельца участка памяти, либо 0000h - свободный участок, 0006h - участок принадлежит драйверу OS XMS UMB 0007h - участок является исключенной верхней памятью драйверов 0008h - участок принадлежит MS DOS FFFAh - участок занят управляющим блоком 386MAX UMB FFFDh - участок заблокирован 386MAX FFFEh - участок принадлежит 386MAX UMB
03h	2	Размер участка в параграфах
05h	3	Зарезервирован
08h	8	"SC" - если участок принадлежит MS DOS, то в нем системный код "SD" - если участок принадлежит MS DOS, то в нем системные данные

По сегментному адресу и размеру участка памяти, контролируемого этим MCB можно определить местоположение следующего MCB в списке.

Адрес первого MCB хранится во внутренней структуре MS DOS, называемой "List of Lists" (список списков). Доступ к указателю на эту структуру можно получить используя функцию 52h "Get List of Lists" int 21h. В результате выполнения этой функции ES:BX будет указывать на список списков. Слово по адресу ES:[BX-2] и есть адрес самого первого MCB.

Размер расширенной памяти находится в ячейках 30h, 31h CMOS. CMOS это энергонезависимая память, в которой хранится информация о конфигурации ПЭВМ. Объем памяти составляет 64 байта. Размер расширенной памяти в Кбайтах можно определить обращаясь к ячейкам CMOS следующим образом:

```
mov  AL,30h ; запись адреса ячейки CMOS
out 70h,AL
in AL,71h   ; чтение младшего байта
mov BL,AL   ; размера расширенной памяти
mov AL,31h  ; запись адреса ячейки CMOS
out 70h,AL
in AL,71h   ; чтение старшего байта
            ; размера расширенной памяти
```

Контрольные вопросы по лабораторной работе №3

1) Что означает "доступный объем памяти"?
2) Где MCB блок Вашей программы в списке?
3) Какой размер памяти занимает программа в каждом случае?

ЛАБОРАТОРНАЯ РАБОТА № 4
«Обработка стандартных прерываний»

Цель работы: В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определенные вектора прерываний. Вектор прерываний хранит адрес подпрограммы обработчика прерываний. При возникновении прерывания, аппаратура компьютера передает управление по соответствующему адресу вектора прерывания. Обработчик прерываний получает управление и выполняет соответствующие действия.

В лабораторной работе № 4 предлагается построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определенные интервалы времени и, при возникновении такого сигнала, возникает прерывание с определенным значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

Порядок выполнения работы

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа **.EXE,** который выполняет следующие функции:

1) Проверяет, установлено ли пользовательское прерывание с вектором 1Ch.
2) Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляется выход по функции 4Ch прерывания int 21h.
3) Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.
4) Выгрузка прерывания по соответствующему значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождении памяти, занимаемой резидентом. Затем осуществляется выход по функции 4Ch прерывания int 21h.

Для того, чтобы проверить установку прерывания, можно поступить следующим образом. Прочитать адрес, записанный в векторе прерывания. Предположим, что этот адрес указывает на точку входа в установленный резидент. На определенном, известном смещении в теле резидента располагается сигнатура, некоторый код, который идентифицирует резидент. Сравнив известное значение сигнатуры с реальным кодом, находящимся в резиденте, можно определить, установлен ли резидент. Если значения совпадают, то резидент установлен. Длину кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным.

Программа должна содержать код устанавливаемого прерывания в виде удаленной процедуры. Этот код будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

1) Сохранить значения регистров в стеке при входе и восстановить их при выходе.
2) При выполнении тела процедуры накапливать общее суммарное число прерываний и выводить на экран. Для вывода на экран следует использовать прерывание int 10h, которое позволяет непосредственно выводить информацию на экран.

Шаг 2. Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания 1Ch установлен. Работа прерывания должна отображаться на экране, а также необходимо проверить размещение прерывания в памяти. Для этого запустите программу ЛР 3, которая отображает карту памяти в виде списка блоков МСВ. Полученные результаты поместите в отчет.

Шаг 3. Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

Шаг 4. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 5. Ответьте на контрольные вопросы.

Необходимые сведения для составления программы

Резидентные обработчики прерываний - это программные модули, которые вызываются при возникновении прерываний определенного типа (сигнал таймера, нажатие клавиши и т.д.), которым соответствуют определенные вектора прерывания. Когда вызывается прерывание, процессор переключается на выполнение кода обработчика, а затем возвращается на выполнение прерванной программы. Адрес возврата в прерванную программу (CS:IP) запоминается в стеке вместе с регистром флагов. Затем в CS:IP загружается адрес точки входа программы обработки прерывания и начинает выполняться его код. Обработчик прерывания должен заканчиваться инструкцией IRET (возврат из прерывания).

Вектор прерывания имеет длину 4 байта. В первом хранится значение IP, во втором - CS. Младшие 1024 байта памяти содержат 256 векторов. Вектор для прерывания 0 начинается с ячейки 0000:0000, для прерывания 1 – с ячейки 0000:0004 и т.д.

Обработчик прерывание - это отдельная процедура, имеющая следующую структуру:

```
ROUT PROC FAR
     PUSH AX   ; сохранение изменяемых регистров
      ...
     <действия по обработке прерывания>
     POP  AX   ; восстановление регистров
      ...
     MOV  AL, 20H
     OUT  20H,AL
     IRET
ROUT ENDP
```

Две последние строки необходимы для разрешения обработки прерываний с более низкими уровнями, чем только что обработанное. Для установки написанного прерывания в поле векторов прерываний используется функция 25H прерывания 21H, которая устанавливает вектор прерывания на указанный адрес.

```
PUSH DS
MOV  DX, OFFSET ROUT ; смещение для процедуры в DX
MOV  AX, SEG ROUT    ; сегмент процедуры
MOV  DS, AX          ; помещаем в DS
MOV  AH, 25H         ; функция установки вектора
MOV  AL, 1CH         ; номер вектора
INT  21H             ; меняем прерывание
POP  DS
```

Программа, выгружающая обработчик прерываний должна восстанавливать оригинальные векторы прерываний. Функция 35 прерывания 21H позволяет восстановить значение вектора прерывания, помещая значение сегмента в ES, а смещение в BX. Программа должна содержать следующие инструкции:

```
; -- хранится в обработчике прерываний
     KEEP_CS DW 0 ; для хранения сегмента
     KEEP_IP DW 0 ; и смещения прерывания
```



```
; -- в программе при загрузке обработчика прерывания
      MOV  AH, 35H   ; функция получения вектора
      MOV  AL, 1CH   ; номер вектора
      INT  21H
      MOV  KEEP_IP, BX  ; запоминание смещения
      MOV  KEEP_CS, ES  ; и сегмента

; -- в программе при выгрузке обработчика прерываний
      CLI
      PUSH DS
      MOV  DX, KEEP_IP
      MOV  AX, KEEP_CS
      MOV  DS, AX
      MOV  AH, 25H
      MOV  AL, 1CH
      INT  21H          ; восстанавливаем вектор
      POP  DS
      STI
```

Для того, чтобы оставить процедуру прерывания резидентной в памяти, следует воспользоваться функцией DOS 31h прерывания 21h. Эта функция оставляет память, размер которой указывается в качестве параметра, занятой, а остальную память освобождает и осуществляет выход в DOS.

Функция 31h int 21h использует следующие параметры:

```
AH – номер функции 31h;
AL – код завершения программы;
DX – размер памяти в параграфах, требуемый резидентной
программе.
```

```
Пример обращения к функции:
mov DX,offset LAST_BYTE ; размер в байтах от начала
сегмента
mov CL,4                ; перевод в параграфы
shr DX,CL
inc DX                  ; размер в параграфах
mov AH,31h
int 21h
```

Вывод на экран информации обработчиком прерываний осуществляется с помощью функций прерывания 10h.

```
;функция вывода символа из AL
outputAL proc
         push ax
         push bx
         push cx
         mov ah,09h   ;писать символ с текущей позиции курсора
         mov bh,0     ;номер видео страницы
         mov cx,1     ;число экземпляров символа для записи
         int 10h      ;выполнить функцию
         pop cx
         pop bx
         pop ax
         ret
;
;
```

```
;функция вывода строки по адресу ES:BP на экран
outputBP proc
         push ax
         push bx
         push dx
         push CX
          mov ah,13h ; функция
          mov al,1 ; sub function code
; 1 = use attribute in BL; leave cursor at end of string
          mov bh,0 ; видео страница
          mov dh,22 ; DH,DL = строка, колонка (считая от 0)
          mov dl,0
          int 10h
          pop CX
          pop dx
          pop bx
          pop ax
          ret
outputBP  endp
;
; Установка позиции курсора
; установка на строку 25 делает курсор невидимым
setCurs proc
         push ax
         push bx
         push dx
         push CX
          mov ah,02h
          mov bh,0
          mov dh,22 ; DH,DL = строка, колонка (считая от 0)
          mov dl,0
          int 10h ; выполнение.
          pop CX
          pop dx
          pop bx
          pop ax
          ret
;
; 03H читать позицию и размер курсора
;    вход:  BH = видео страница
;   выход:  DH,DL = текущие строка, колонка курсора
;           CH,CL = текущие начальная, конечная строки курсора
getCurs  proc
         push ax
         push bx
         push dx
         push CX
          mov ah,03h
          mov bh,0
          int 10h ; выполнение.
; выход:  DH,DL = текущие строка, колонка курсора
;         CH,CL = текущие начальная, конечная строки курсора
          pop CX
```

```
        pop dx
        pop bx
        pop ax
        ret
```

Контрольные вопросы по лабораторной работе №4

1) Как реализован механизм прерывания от часов?
2) Какого типа прерывания использовались в работе?

ЛАБОРАТОРНАЯ РАБОТА № 5
«Сопряжение стандартного и пользовательского обработчиков прерываний»

Цель работы: Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерывания получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывает скан-код и осуществляет определенные действия, если скан-код совпадает с определенными кодами, которые он должен обрабатывать. Если скан-код не совпадает с этими кодами, то управление передается стандартному прерыванию.

Порядок выполнения работы

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа **.EXE,** который выполняет такие же функции, как в программе ЛР 4, а именно:

1) Проверяет, установлено ли пользовательское прерывание с вектором 09h.
2) Если прерывание не установлено то, устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний. Адрес точки входа в стандартный обработчик прерывания находится в теле пользовательского обработчика. Осуществляется выход по функции 4Ch прерывания int 21h.
3) Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.

Выгрузка прерывания по соответствующему значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождении памяти, занимаемой резидентом. Затем осуществляется выход по функции 4Ch прерывания int 21h.

Для того чтобы проверить установку прерывания, можно поступить следующим образом. Прочитать адрес, записанный в векторе прерывания. Предположим, что этот адрес указывает на точку входа в установленный резидент. На определенном, известном смещении в теле резидента располагается сигнатура, некоторый код, который идентифицирует резидент. Сравнив известное значение сигнатуры с реальным кодом, находящимся в резиденте, можно определить, установлен ли резидент. Если значения совпадают, то резидент установлен. Длину кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным.

Программа должна содержать код устанавливаемого прерывания в виде удаленной процедуры. Этот код будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

1) Сохранить значения регистров в стеке при входе и восстановить их при выходе.
2) При выполнении тела процедуры анализируется скан-код.
3) Если этот код совпадает с одним из заданных, то требуемый код записывается в буфер клавиатуры.
4) Если этот код не совпадает ни с одним из заданных, то осуществляется передача управления стандартному обработчику прерывания.

Шаг 2. Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания 09h установлен. Работа прерывания проверяется введением различных символов, обрабатываемых установленным обработчиком и стандартным обработчиком.

Шаг 3. Также необходимо проверить размещение прерывания в памяти. Для этого запустите программу ЛР 3, которая отображает карту памяти в виде списка блоков МСВ. Полученные результаты поместите в отчет.

Шаг 4. Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

Шаг 5. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 6. Ответьте на контрольные вопросы.

Необходимые сведения для составления программы

Клавиатура содержит микропроцессор, который воспринимает каждое нажатие на клавишу и посылает скан-код в порт микросхемы интерфейса с периферией. Когда скан-код поступает в порт, то вызывается аппаратное прерывание клавиатуры (int 09h). Процедура обработки этого прерывания считывает номер клавиши из порта 60h, преобразует номер клавиши в соответствующий код, выполняет установку флагов в байтах состояния, загружает номер клавиши и полученный код в буфер клавиатуры.

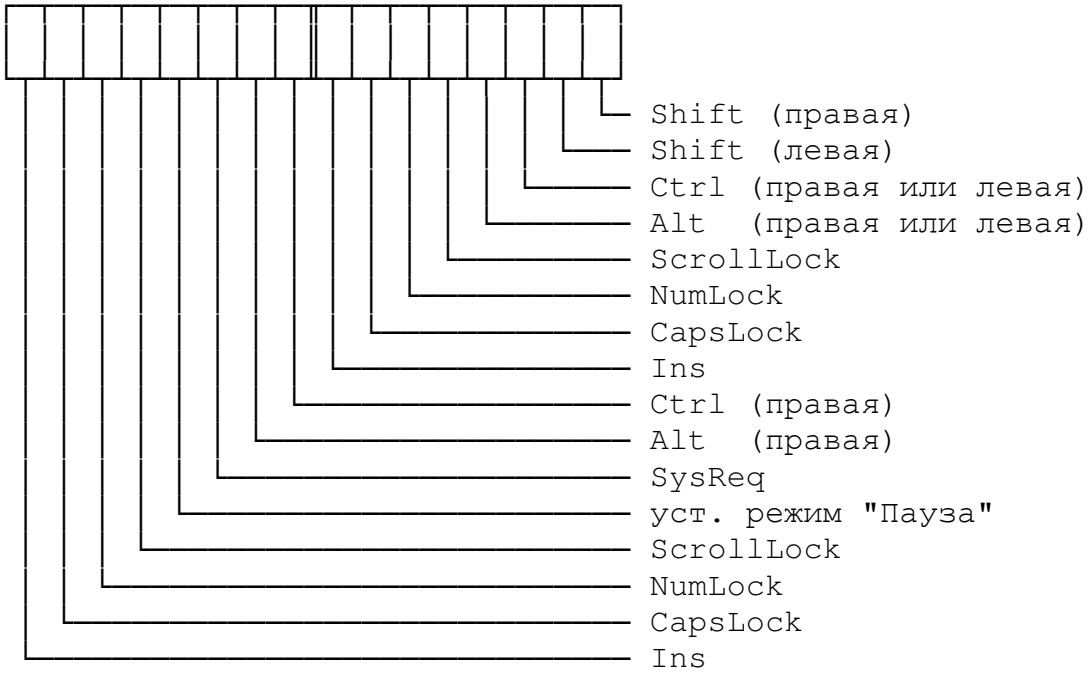
В прерывании клавиатуры можно выделить три основных шага:

1. Прочитать скан-код и послать клавиатуре подтверждающий сигнал.
2. Преобразовать скан-код в номер кода или в установку регистра статуса клавиш-переключателей.
3. Поместить код клавиши в буфер клавиатуры.

Текущее содержимое буфера клавиатуры определяется указателями на начало и конец записи. Расположение в памяти необходимых данных представлено в таблице.

Адрес в памяти	Размер в байтах	Содержимое
0040:001A	2	Адрес начала буфера клавиатуры
0040:001C	2	Адрес конца буфера клавиатуры
0040:001E	32	Буфер клавиатуры
0040:0017	2	Байты состояния

Флаги в байтах состояния устанавливаются в 1, если нажата соответствующая клавиша или установлен режим. Соответствие флагов и клавиш показано ниже.



В момент вызова прерывания скан-код будет находиться в порте 60h. Поэтому сначала надо этот код прочитать командой IN и сохранить на стеке. Затем используется

порт 61H, чтобы быстро послать сигнал подтверждения микропроцессору клавиатуры. Надо просто установить бит 7 в 1, а затем сразу изменить его назад в 0. Заметим, что бит 6 порта 61H управляет сигналом часов клавиатуры. Он всегда должен быть установлен в 1, иначе клавиатура будет выключена. Эти адреса портов применимы и к AT, хотя он и не имеет микросхемы интерфейса с периферией 8255.

Сначала скан-код анализируется на предмет того, была ли клавиша нажата (код нажатия) или отпущена (код освобождения). Код освобождения состоит из двух байтов: сначала 0F0H, а затем скан-код. Все коды освобождения отбрасываются, кроме случая клавиш-переключателей, для которых делаются соответствующие изменения в байтах их статуса. С другой стороны, все коды нажатия обрабатываются. При этом опять могут изменяться байты статуса клавиш-переключателей. В случае же символьных кодов, надо проверять байты статуса, чтобы определить, например, что скан-код 30 соответствует нижнему или верхнему регистру буквы A. После того как введенный символ идентифицирован, процедура ввода с клавиатуры должна найти соответствующий ему код ASCII или расширенный код. Приведенный пример слишком короток, чтобы рассмотреть все случаи. В общем случае скан-коды сопоставляются элементам таблицы данных, которая анализируется инструкцией XLAT. XLAT принимает в AL число от 0 до 255, а возвращает в AL 1-байтное значение из 256-байтной таблицы, на которую указывает DS:BX. Таблица может находиться в сегменте данных. Если в AL находился скан-код 30, то туда будет помещен из таблицы байт номер 30 (31-й байт, так как отсчет начинается с нуля). Этот байт в таблице должен быть установлен равным 97, давая код ASCII для "a". Конечно для получения заглавной A нужна другая таблица, к которой обращение будет происходить, если статус сдвига установлен. Или заглавные буквы могут храниться в другой части той же таблицы, но в этом случае к скан-коду надо будет добавлять смещение, определяемое статусом клавиш-переключателей.

Номера кодов должны быть помещены в буфер клавиатуры. Процедура должна сначала проверить, имеется ли в буфере место для следующего символа. Буфер устроен как циклическая очередь. Ячейка памяти 0040:001A содержит указатель на голову буфера, а 0040:001C - указатель на хвост. Эти словные указатели дают смещение в области данных BIOS (которая начинается в сегменте 40H) и находятся в диапазоне от 30 до 60. Новые символы вставляются в ячейки буфера с более старшими адресами, а когда достигнута верхняя граница, то следующий символ переносится в нижний конец буфера. Когда буфер полон, то указатель хвоста на 2 меньше указателя на голову - кроме случая, когда указатель на голову равен 30 (начало области буфера), а в этом случае буфер полон, когда указатель хвоста равен 60. Для вставки символа в буфер, надо поместить его в позицию, на которую указывает хвост буфера и затем увеличить указатель хвоста на 2; если указатель хвоста был равен 60, то надо изменить его значение на 30.

Код для отработки прерывания 09H

```
      push    ax
      in      al,60H      ;читать ключ
      cmp     al,REQ_KEY  ;это требуемый код?
      je    do_req      ; да, активизировать обработку REQ_KEY
                         ;  нет, уйти на исходный обработчик
      pop   ax
      jmp   cs:[int9_vect] ;переход на первоначальный обработчик
do_req:
;следующий код необходим для отработки аппаратного прерывания
      in  al,61H   ;взять значение порта управления клавиатурой
      mov ah,al     ; сохранить его
      or  al,80h    ;установить бит разрешения для клавиатуры
      out 61H,al    ; и вывести его в управляющий порт
      xchg ah,al    ;извлечь исходное значение порта
      out 61H,al    ;и записать его обратно
```



```
        mov     al,20H      ;послать сигнал "конец прерывания"
        out     20H,al      ; контроллеру прерываний 8259
        ;------ дальше – прочие проверки
```

Записать символ в буфер клавиатуры можно с помощью функции 05h прерывания 16h:

```
     mov  ah,05h  ; Код функции
     mov  cl,'D' ; Пишем символ в буфер клавиатуры
     mov  ch,00h ;
     int  16h ;
     or al,al ; проверка переполнения буфера
     jnz skip ; если переполнен идем skip
; работать дальше
skip:  ; очистить буфер и повторить
```

Контрольные вопросы по лабораторной работе №5

1) Какого типа прерывания использовались в работе?
2) Чем отличается скан код от кода ASCII?

ЛАБОРАТОРНАЯ РАБОТА № 6
«Построение модуля динамической структуры»

Цель работы: Исследование возможности построения загрузочного модуля динамической структуры. В отличии от предыдущих лабораторных работ в этой работе рассматривается приложение, состоящее из нескольких модулей, а не из одного модуля простой структуры. В этом случае разумно предположить, что все модули приложения находятся в одном каталоге и полный путь в этот каталог можно взять из среды, как это делалось в работе 2. Понятно, что такое приложение должно запускаться в соответствии со стандартами ОС.

В работе исследуется интерфейс между вызывающим и вызываемым модулями по управлению и по данным. Для запуска вызываемого модуля используется функция 4B00h прерывания int 21h. Все загрузочные модули находятся в одном каталоге. Необходимо обеспечить возможность запуска модуля динамической структуры из любого каталога.

Порядок выполнения работы

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа **.EXE,** который выполняет функции:

1) Подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором находится он сам. Вызываемому модулю передается новая среда, созданная вызывающим модулем и новая командная строка.
2) Вызываемый модуль запускается с использованием загрузчика.
3) После запуска проверяется выполнение загрузчика, а затем результат выполнения вызываемой программы. Необходимо проверять причину завершения и, в зависимости от значения, выводить соответствующее сообщение. Если причина завершения 0, то выводится код завершения.

В качестве вызываемой программы необходимо взять программу ЛР 2, которая распечатывает среду и командную строку. Эту программу следует немного модифицировать, вставив перед выходом из нее обращение к функции ввода символа с клавиатуры. Введенное значение записывается в регистр AL и затем происходит обращение к функции выхода 4Ch прерывания int 21h.

Шаг 2. Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.

Введите произвольный символ из числа A-Z. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

Шаг 3. Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.

Введите комбинацию символов Ctrl-C. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

Шаг 4. Запустите отлаженную программу, когда текущим каталогом является какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули.

Повторите ввод комбинаций клавиш. Занесите полученные данные в отчет.

Шаг 5. Запустите отлаженную программу, когда модули находятся в разных каталогах. Занесите полученные данные в отчет.

Необходимые сведения для составления программы

Для загрузки и выполнения одной программы из другой используется функция 4B00h прерывания int 21h (загрузчик ОС). Перед обращением к этой функции необходимо выполнить следующие действия:

1) Подготовить место в памяти. При начальном запуске программы ей отводится вся доступная в данный момент память OS, поэтому необходимо освободить место в памяти. Для этого можно использовать функцию 4Ah прерывания int 21h. Эта функция позволяет уменьшить отведенный программе блок памяти. Перед вызовом функции надо определить объем памяти, необходимый программе ЛР6 и задать в регистре BX число параграфов, которые будут выделяться программе. Если функция 4Ah не может быть выполнена, то устанавливается флаг переноса CF=1 и в AX заносится код ошибки:

7 - разрушен управляющий блок памяти;

8 - недостаточно памяти для выполнения функции;

9 - неверный адрес блока памяти.

Поэтому после выполнения каждого прерывания int 21h следует проверять флаг переноса CF=1.

2) Создать блок параметров. Блок параметров - это 14-байтовый блок памяти, в который помещается следующая информация:

```
dw  сегментный адрес  среды
dd  сегмент  и  смещение командной  строки
dd  сегмент  и  смещение первого FCB
dd  сегмент и смещение второго FCB
```

Если сегментный адрес среды 0, то вызываемая программа наследует среду вызывающей программы. В противном случае вызывающая программа должна сформировать область памяти в качестве среды, начинающуюся с адреса кратного 16 и поместить этот адрес в блок параметров.

Командная строка записывается в следующем формате:

первый байт - счетчик, содержащий число символов в командной строке, затем сама командная строка, содержащая не более 128 символов.

На блок параметров перед загрузкой вызываемой программы должны указывать ES:BX.

3) Подготовить строку, содержащую путь и имя вызываемой программы. В конце строки должен стоять код ASCII 0. На подготовленную строку должны указывать DS:DX.

4) Сохранить содержимое регистров SS и SP в переменных. При восстановлении SS и SP нужно учитывать, что DS необходимо также восстановить.

Когда вся подготовка выполнена, вызывается загрузчик OS следующей последовательностью команд:

```
mov   AX,4B00h
int   21h
```

Если вызываемая программа не была загружена, то устанавливается флаг переноса CF=1 и в AX заносится код ошибки:

- 1 - если номер функции неверен;
- 2 - если файл не найден;
- 5 - при ошибке диска;
- 8 - при недостаточном объеме памяти;
- 10 - при неправильной строке среды;
- 11 - если не верен формат.

Если CF=0, то вызываемая программа выполнена и следует обработать ее завершение. Для этого необходимо воспользоваться функцией 4Dh прерывания int 21h. В качестве результата функция возвращает в регистре AH причину, а в регистре AL код завершения.

Причина завершения в регистре AH представляется следующими кодами:

- 0 - нормальное завершение;
- 1 - завершение по Ctrl-Break;
- 2 - завершение по ошибке устройства;
- 3 - завершение по функции 31h, оставляющей программу резидентной.

Код завершения формируется вызываемой программой в регистре AL перед выходом в OS с помощью функции 4Ch прерывания int 21h.

В качестве вызываемой программы целесообразно использовать программу, разработанную в Лабораторной работе №2, модифицировав ее следующим образом. Перед выходом из программы перед выполнением функции 4Ch прерывания int 21h следует запросить с клавиатуры символ и поместить введенный символ в регистр AL, в качестве кода завершения. Это можно сделать с помощью функции 01h прерывания int 21h.

```
mov   AH,01h
int   21h
```

Введенный символ остается в регистре AL и служит аргументом для функции 4Ch прерывания int 21h.

Контрольные вопросы по лабораторной работе №6

1) Как реализовано прерывание Ctrl-C?
2) В какой точке заканчивается вызываемая программа, если код причины завершения 0?
3) В какой точке заканчивается вызываемая программа по прерыванию Ctrl-C?

ЛАБОРАТОРНАЯ РАБОТА № 7
«Построение модуля оверлейной структуры»

Цель работы: Исследование возможности построения загрузочного модуля оверлейной структуры. Исследуется структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов. Для запуска вызываемого оверлейного модуля используется функция 4B03h прерывания int 21h. Все загрузочные и оверлейные модули находятся в одном каталоге.

В этой работе также рассматривается приложение, состоящее из нескольких модулей, поэтому все модули помещаются в один каталог и вызываются с использованием полного пути.

Порядок выполнения работы

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа **.EXE,** который выполняет функции:

1) Освобождает память для загрузки оверлеев.
2) Читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки.
3) Файл оверлейного сегмента загружается и выполняется.
4) Освобождается память, отведенная для оверлейного сегмента.
5) Затем действия 1)-4) выполняются для следующего оверлейного сегмента.

Шаг 2. Также необходимо написать и отладить оверлейные сегменты. Оверлейный сегмент выводит адрес сегмента, в который он загружен.

Шаг 3. Запустите отлаженное приложение. Оверлейные сегменты должны загружаться с одного и того же адреса, перекрывая друг друга.

Шаг 4. Запустите приложение из другого каталога. Приложение должно быть выполнено успешно.

Шаг 5. Запустите приложение в случае, когда одного оверлея нет в каталоге. Приложение должно закончиться аварийно.

Шаг 6. Занесите полученные результаты в виде скриншотов в отчет. Оформите отчет в соответствии с требованиями.

Необходимые сведения для составления программы

Для организации программы, имеющей оверлейную структуру, используется функция 4B03h прерывания int 21h. Эта функция позволяет в отведенную область памяти, начинающуюся с адреса сегмента, загрузить программу, находящуюся в файле на диске. Передача управления загруженной программе этой функцией не осуществляется и префикс сегмента программы (PSP) не создается. Обращение к функции 4B03h:

```
AX=4B03h – код функции;
DS:DX – указывает на строку ASCIIZ,  содержащую путь к оверлею;
ES:BX –  указатель на блок параметров,  который представляет
         собой два слова памяти, содержащих сегментный адрес
         загрузки программы.
```

Если флаг переноса CF=1 после выполнения функции, то произошли ошибки и регистр AX содержит код ошибки. Значение регистра AX характеризует следующие ситуации:

- 1 - несуществующая функция;
- 2 - файл не найден;
- 3 - маршрут не найден;
- 4 - слишком много открытых файлов;
- 5 - нет доступа;
- 8 - мало памяти;

- 10 - неправильная среда.

Если флаг переноса CF=0, то оверлей загружен в память.

Перед загрузкой оверлея вызывающая программа должна освободить память по функции 4Ah прерывания int 21h. Затем определить размер оверлея. Это можно сделать с помощью функции 4Eh прерывания 21h. Перед обращением к функции необходимо определить область памяти размером в 43 байта под буфер DTA, которую функция заполнит, если файл будет найден.

Функция использует следующие параметры:

CX - значение байта атрибутов, которое для файла имеет значение 0;

DS:DX - указатель на путь к файлу, который записывается в формате строки ASCIIZ.

Если флаг переноса CF=1 после выполнения функции, то произошли ошибки и регистр AX содержит код ошибки. Значение регистра AX характеризует следующие ситуации:

- 2 - файл не найден;
- 3 - маршрут не найден.

Если CF=0, то в области памяти буфера DTA со смещением 1Ah будет находится младшее слово размера файла, а в слове со смещением 1Ch - старшее слово размера памяти в байтах.

Полученный размер файла следует перевести в параграфы, причем следует взять большее целое числа параграфов. Затем необходимо отвести память с помощью функции 48h прерывания 21h. После этого необходимо сформировать параметры для функции 4B03h и выполнить ее.

После отработки оверлея необходимо освободить память с помощью функции 49h прерывания int 21h. Обращение к этой функции содержит следующие параметры:

```
AH=49h – код функции;
ES – сегментный адрес освобождаемой памяти.
```

Оверлейный сегмент не является загрузочным модулем типов **.COM** или **.EXE**. Он представляет собой кодовый сегмент, который оформляется в ассемблере как функция с точкой входа по адресу 0 и возврат осуществляется командой RETF. Это необходимо сделать, потому что возврат управления должен быть осуществлен в программу, выполняющую оверлейный сегмент. Если использовать функции выхода 4Ch прерывания int 21h, то программа закончит свою работу.

Контрольные вопросы по лабораторной работе №7

1) Как должна быть устроена программа, если в качестве оверлейного сегмента использовать **.COM** модули?

Литература

Р. Джордейн. Справочник программиста. - М., «Финансы», 1991г.

Рудаков П.И, Финогенов К.Г. Язык ассемблера: уроки программирования. – М.; «Диалог-МИФИ», 2001, 640с.

http://www.cyberdengi.com/FoundationsOfAssembler/Theme01.html

http://www.codenet.ru/progr/dos/int_0009.php