МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Курсовая работа

«Утилита анализа кода программы»

по дисциплине

«Принципы и методы организации системных

программных средств»

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кочешков А. А

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сухоруков В. А

(подпись) (фамилия, и.,о.)

19-В-2\_\_\_\_\_

(шифр группы)

Работа защищена «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2022

Оглавление

[Цель работы 3](#_Toc105230579)

[Ход работы 3](#_Toc105230580)

[Теоретическая часть 3](#_Toc105230581)

[Отладчик ассемблерного кода - Turbo Debugger 5](#_Toc105230582)

[Алгоритм анализа кода exe-файла 7](#_Toc105230583)

[Словесное описание 7](#_Toc105230584)

[Блок-схема 8](#_Toc105230585)

[Код программы 9](#_Toc105230586)

[Проверка работы программы 11](#_Toc105230587)

[Оценка качества работы алгоритма 13](#_Toc105230588)

[Вывод 13](#_Toc105230589)

[Список источников информации 14](#_Toc105230590)

[Приложения 15](#_Toc105230591)

[Приложение 1 – исходный код программы TEST\_PROG1 15](#_Toc105230592)

[Приложение 2 – исходный код программы TEST\_PROG2 16](#_Toc105230593)

# Цель работы

Разработать программу анализа кода в exe-файле программы для выявления вызовов прерываний и команд ввода-вывода (in, out). Обработанный отчет о номерах вызываемых прерываний и портов ввода-вывода выводить на экран и в log-файл.

# Ход работы

## Теоретическая часть

Почти все задачи, решаемые программами, нуждаются в данных, которые вырабатываются в результате решения другой задачи, или передаются от внешних устройств. Для общения программ между собой и взаимодействия с внешними устройствами используются механизмы прерываний и команды ввода-вывода.

Прерывание (interrupt) – событие, требующие немедленной реакции со стороны процессора. Реакция состоит в том, что процессор прерывает обработку текущей программы (прерываемой программы) и переходит к выполнению некоторой другой программы (прерывающей программы), специально предназначенной для данного события. По завершении этой программы процессор возвращается к выполнению прерванной программы.

Каждое событие, требующее прерывания, сопровождается сигналом прерывания, оповещающим об этом вычислительную машину, и называемым запросом прерывания.

Состояние программы представляет собой совокупность состояний всех запоминающих элементов в соответствующий момент времени (например, после выполнения последней команды). При возникновении прерывания ЭВМ сохраняет в стеке содержимое счетчика команд и загружает в него адрес соответствующего вектора прерывания. Последней командой подпрограммы обработки прерывания должна быть команда, которая осуществляет возврат в основную программу и восстановление предварительно сохраненного счетчика команд.

Команда INT вызывает обработчик указанного операндом прерывания (константой). Операнд определяет номер вектора системного прерывания BIOS от 0 до 255, представленный в виде беззнакового 8-битного целого числа. При вызове обработчика прерывания в стеке сохраняются регистры EIP, CS и EFLAGS.

|  |  |  |
| --- | --- | --- |
| **Команда** | **Пояснение** | **Описание** |
| INT с | EIP → стек CS → стек EFLAGS → стек переход к вектору c | Программное прерывание |

Таблица 1. Формат команды INT

Вводом/выводом (ВВ) называется передача данных между ядром ЭВМ, включающим в себя микропроцессор и основную память, и внешними устройствами.

Подключение внешних устройств к системной шине осуществляется посредством электронных схем, называемых контроллерами ВВ (интерфейсами ВВ). Они согласуют уровни электрических сигналов, а также преобразуют машинные данные в формат, необходимый устройству, и наоборот. Обычно контроллеры ВВ конструктивно оформляются вместе с процессором в виде интерфейсных плат.

В процессе ввода/вывода передается информация двух видов: управляющие данные (слова) и собственно данные, или данные-сообщения. Управляющие данные от процессора, называемые также командными словами или приказами, инициируют действия, не связанные непосредственно с передачей данных, например запуск устройства, запрещение прерываний и т.п. Управляющие данные от внешних устройств называются словами состояния; они содержат информацию об определенных признаках, например о готовности устройства к передаче данных, о наличии ошибок при обмене и т.п. Состояние обычно представляется в декодированной форме - один бит для каждого признака.

Регистр, содержащий группу бит, к которой процессор обращается в операциях ВВ, образует порт ВВ. Таким образом, наиболее общая программная модель внешнего устройства, которое может выполнять ввод и вывод, содержит четыре регистра ВВ: регистр выходных данных (выходной порт), регистр входных данных (входной порт), регистр управления и регистр состояния. Каждый из этих регистров должен иметь однозначный адрес, который идентифицируется дешифратором адреса. В зависимости от особенностей устройства общая модель конкретизируется, например, отдельные регистры состояния и управления объединяются в один регистр, в устройстве ввода (вывода) имеется только регистр входных (выходных) данных, для ввода и вывода используется двунаправленный порт.

Для взаимодействия с портами используются команды IN и OUT, операндами которых являются номер порта и значение.

|  |  |  |  |
| --- | --- | --- | --- |
| **Команда** | **Операнды** | **Пояснение** | **Описание** |
| IN | AL, c AX, c EAX, c AL, DX AX, DX EAX, DX | AL= port byte AX= port word EAX= port dword AL= [DX-port] AX= [DX-port] EAX= [DX-port] | Ввод из порта |
| OUT | c, AL c, AX c, EAX DX, AL DX, AX DX, EAX | port byte=AL port word=AX port dword=EAX [DX-port]=AL [DX-port]=AX [DX-port]=EAX | Вывод в порт |

Таблица 2. Формат команд IN и OUT

Так же существуют специальные команды ввода вывода:

|  |  |
| --- | --- |
| Команда | Описание |
| OUTSB | Вывести байт из ячейки памяти DS:SI в порт в/в по адресу в DX |
| OUTSW | Вывести слово из ячейки памяти DS:SI в порт в/в по адресу в DX |
| OUTSD | Вывести двойное слово из ячейки памяти DS:SI в порт в/в по адресу в DX |
| INSB | Загрузить байт из порта в/в DX в память по адресу ES:DI |
| INSW | Загрузить слово из порта в/в DX в память по адресу ES:DI |
| INSD | Загрузить двойное слово из порта в/в DX в память по адресу ES:DI |

Таблица 3. Специальные команды ввода/вывода

## Отладчик ассемблерного кода - Turbo Debugger

Отладка (debugging) — один из важнейших этапов разработки программного обеспечения (английский термин bug означает "ошибка в программе"). В процессе отладки путем детального анализа в компьютерных программах выявляются и устраняются возможные логические ошибки, которые не обнаруживаются на стадии компиляции.

Отладчики (debugger) — это вспомогательные программы (утилиты), включаемые в набор инструментальных средств программиста для выполнения отладки других программ. Отладчики предоставляют программисту возможность выполнять программу по шагам, следить за изменениями данных и проверять выполнение условий. В зависимости от уровня языка, которым оперирует отладчик, можно выделить два их типа.

Отладчики исходного кода дают программисту возможность видеть текст программы на языке высокого уровня (например, Си), проверять значения отдельных переменных и агрегатов данных (таких, как массивы), используя их имена.

Отладчики машинного уровня отслеживают реально выполняемые машинные команды, отображаемые в виде команд ассемблера. Они позволяют также просматривать содержимое ячеек памяти и регистров микропроцессора.

Turbo Debugger — это отладчик второго типа.

Рассмотрим машинное представление команд прерывания и ввода/вывода.

Запишем в поле кода команды int 21h, int 16h, int 0h, int 22h (Рис 1). Команда транслируется в машинный код как последовательность бит. Для лучшего понимания пользователем кодировки команды, Turbo Debugger использует 16-ричный формат отображения. Команды прерывания кодируются как CD<Номер\_прерывания>h (Рис 2).

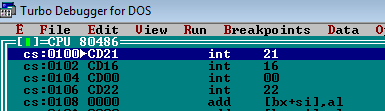


Рис 1. Рис 2.

У каждого типа прерывания есть несколько функций, которые выполняют разные действия. Некоторые функции прерывания 21h представлены в таблице 4, 16h – в таблице 5.

|  |  |
| --- | --- |
| **Номер функции** | **Описание функции** |
| 02h | Вывод одного символа на дисплей |
| 09h | Вывод строки на дисплей |
| 0ah | Ввод строки |
| 27h | Считать блок файла |
| 4ch | Завершить программу |
| 5bh | Создать файл |

Таблица 4. Функции 21h прерывания

|  |  |
| --- | --- |
| **Номер функции** | **Описание функции** |
| 00h | Читать (ожидать) следующую нажатую клавишу |
| 01h | Проверить готовность символа (и показать его если так). |
| 02h | Читать состояние shift-клавиш. |

Таблица 5. Функции 16h прерывания

Номер функции необходимо поместить в регистр ah. Для этого используется команда mov ah,c. Её представление в машинном коде – B4<Константа>h (Рис 3).

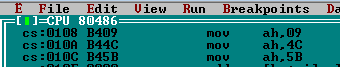


Рис 3.

Теперь рассмотрим представление команд ввода/вывода во всех доступных вариантах (Рис 4, 5). Для команд ввода коды операций – E4, E5, EC, ED, 6C, 6D, 666D; для команд вывода – E6, E7, EE, EF, 6E, 6F, 666F.

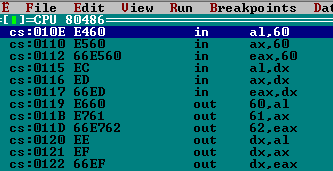




Рис 4. Рис 5.

Исходя из полученных данных, можно составить алгоритм поиска команд прерываний и операций ввода/вывода.

## Алгоритм анализа кода exe-файла

### Словесное описание

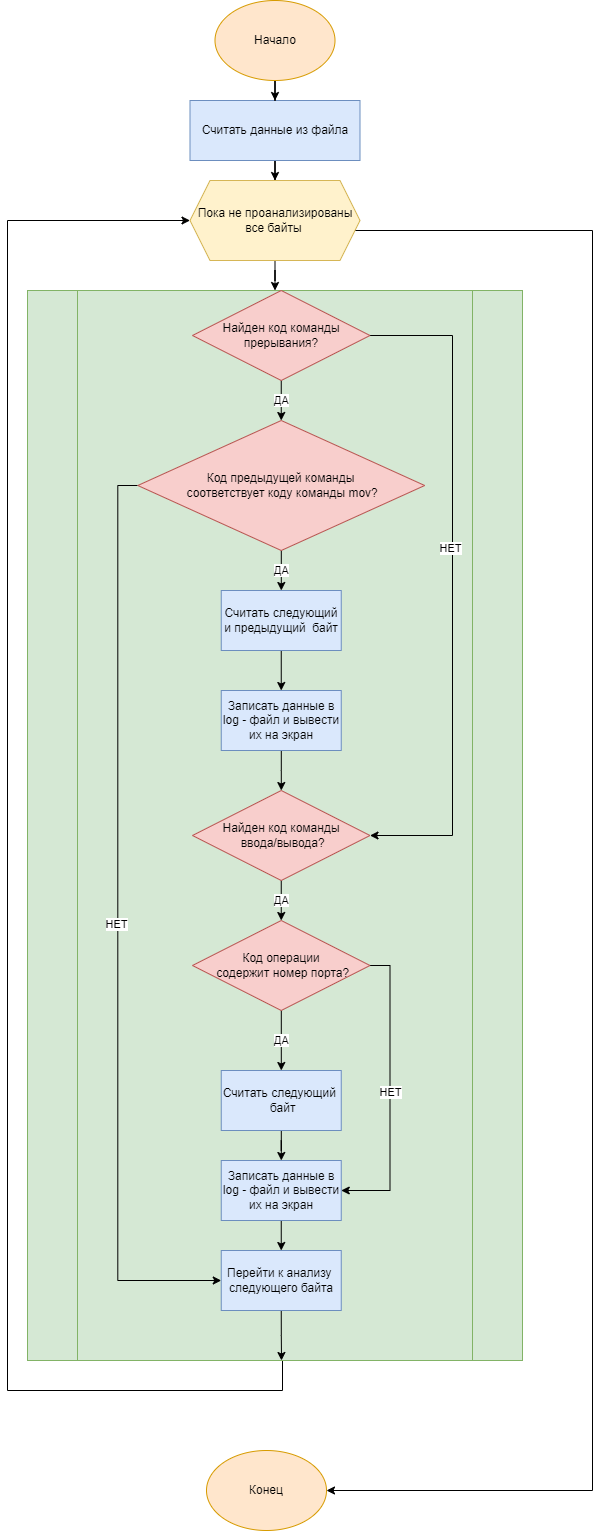
Для поиска вызовов прерывания и операций ввода вывода нужно:

1. Считать данные из файла в бинарном виде.
2. Проанализировать каждый байт на соответствие коду операции ввода/вывода и вызова прерываний.
3. Если найден код команды вызова прерывания:
4. Если предыдущая команда mov:
5. Считать следующий байт для определения номера прерывания
6. Считать предыдущий байт – для определения функции.
7. Вывести данные на экран.
8. Сохранить данные в log-файл.
9. Если найдена команда ввода вывода:
10. Если код операции содержит порт:
11. Считать следующий байт для определения номера порта.
12. Вывести данные на экран.
13. Сохранить данные в log-файл.

Представленный алгоритм не обеспечивает точность нахождения команд вызова прерываний и операций ввода/вывода. Если значение байта равно коду операции искомой команды, то это не означает того, что была найдена нужная команда. Данный код может быть найден в области данных приложения, или являться значением переменной.

Увеличение точности поиска команд прерывания обеспечивается просмотром кода операции предыдущей команды, команде int обычно предшествует команда mov, которая заносит в регистр ax номер функции.

### Блок-схема



### Код программы

Для написания программы-анализатора был выбран язык python, потому что на нём проще и быстрее можно написать небольшую программу. Код языка python более «читабельный», по сравнению с другими языками высокого уровня. Если в дальнейшем потребуется использовать данную программу большим числом пользователей, ей можно будет переписать на другом языке программирования, который оптимизирует программу для наибольшей эффективности.

В языке программирования питон файл можно открыть функций open() с аргументами ‘r’ и ‘b’ для чтения в бинарном виде, считать – функцией read(). Код функции для чтения файла:

def read\_file(file\_name):

f = open (file\_name,'rb')

data=f.read()

return data

Составим функцию анализа кода, согласно алгоритму.

#Поиск вызовов прерываний и операций ввода/вывода

def analize (data):

file=open ('analize.log','w')

#Адрес операции

address=0

#Для всех элементов в массиве

for i in range (0,len(data)):

#Проверка на прерывание

#Если содержимое байта равно 205d(CDh)

if data[i] == 205:

#Если содержимое байта, идущего на 2 раньше,

#равно 180d(B4h)

if data[i-2]==180:

string='Адрес: '+hex(address)+

' Прерывание № ' +

hex(data[i+1])+

' функция - '+

hex(data[i-1])

print(string)

file.write(string)

#Проверка на команды ввода

#Если содержимое байта равно 228d(E4h)или 229d(E5h),

#то найдена команда in с явным указание номера порта

if data [i]== 228 or data [i]== 229:

string='Адрес: '+hex(address)+

' Команда in, номер порта - '+

hex(data[i+1])

print(string)

file.write(string)

#Если содержимое байта равно 236d(ECh)или 237d(EDh),

#то найдена команда in, номер порта находится в регистре dx

if data [i]== 236 or data [i]== 237:

string='Адрес: '+hex(address)+

' Команда in, номер порта указан в регистре dx'

print(string)

file.write(string)

#Если содержимое байта равно 108d(6Сh)

#то найдена команда insb

if data [i]== 108:

string='Адрес: '+hex(address)+' Команда insb'

print(string)

file.write(string)

#Если содержимое байта равно 109d(6Dh)

#то найдена команда insw

if data [i]== 109:

string='Адрес: '+hex(address)+' Команда insw'

print(string)

file.write(string)

#Проверка на команды вывода

#Если содержимое байта равно 230d(E6h)или 232d(E7h),

#то найдена команда in с явным указание номера порта

if data [i]== 230 or data [i]== 231:

string='Адрес: '+hex(address)+

' Команда out, номер порта - '+ hex(data[i+1])

print(string)

file.write(string)

#Если содержимое байта равно 238d(EEh)или 239d(EFh),

#то найдена команда in, номер порта находится в регистре dx

if data [i]== 238 or data [i]== 239:

string='Адрес: '+hex(address)+

' Команда out, номер порта указан в регистре dx'

print(string)

file.write(string)

#Если содержимое байта равно 110d(6Eh)

#то найдена команда outsb

if data [i]== 110:

string='Адрес: '+hex(address)+' Команда outsb'

print(string)

file.write(string)

#Если содержимое байта равно 111d(6Fh)

#то найдена команда outsw

if data [i]== 111:

string='Адрес: '+hex(address)+' Команда outsw'

print(string)

file.write(string)

#Увеличение адреса

address=address+1

Далее составим часть программы, отвечающую за графический интерфейс пользователя, и проверим работоспособность алгоритма .

## Проверка работы программы

Для проверки работоспособности алгоритма создадим программу, содержащую две команды вызова прерываний и четыре команды обращения к портам ввода/вывода (*Исходный код .asm файла приведен в приложении 1).*

На рисунке 6 представлен результат работы программы в графическом режиме, на рисунке 7 представлено содержимое созданного log-файла.

Все «нужные» команды были найдены и занесены в соответствующие таблицы. «Лишних» команд найдено небыло.

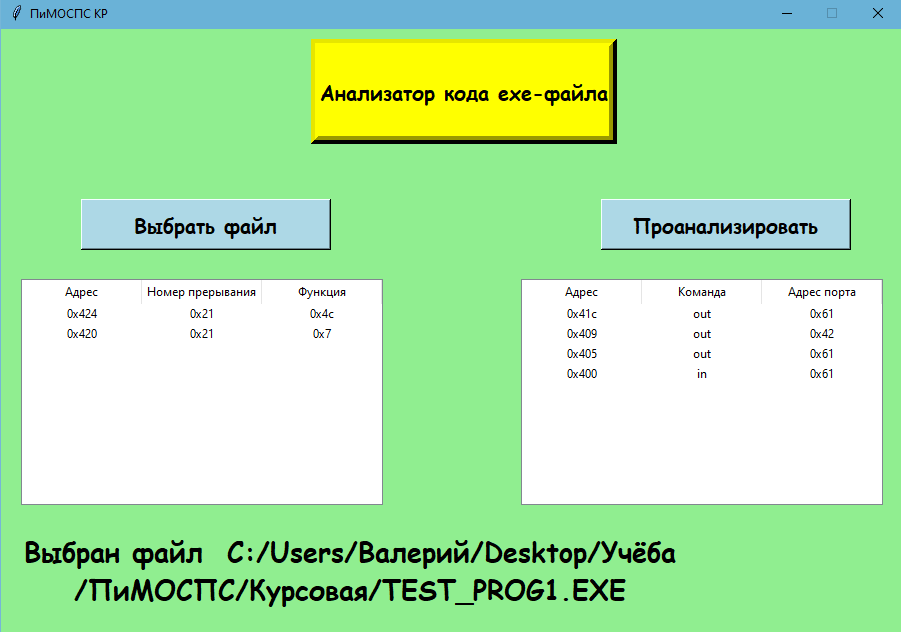


Рис 6. Результат работы программы-анализатора с файлом TEST\_PROG1.exe

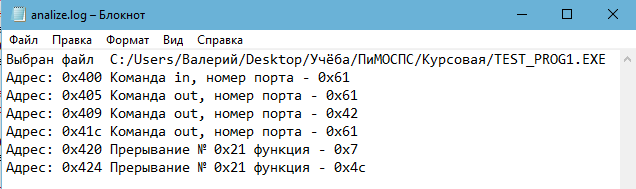


Рис 7. Log-файл, созданный в результате работы программы

Теперь проведем анализ содержимого exe-файла, содержащего шесть команд вызова прерываний, и не содержащего команд ввода/вывода (*Исходный код .asm файла приведен в приложении 2).*

На рисунке 8 представлен результат работы программы. Все команды вызова прерываний были найдены, «лишних» нет.

А в таблице обращений к портам ввода/вывода содержится 2 записи, которых быть недолжно. Это произошло из-за того, что байты по адресам 522h и 201h содержат данные, совпадающие с кодом операции В/В.

Избежать этой ошибки можно только, если анализировать всё содержимое файла, и «пытаться составлять команды» - написать «свой отладчик».

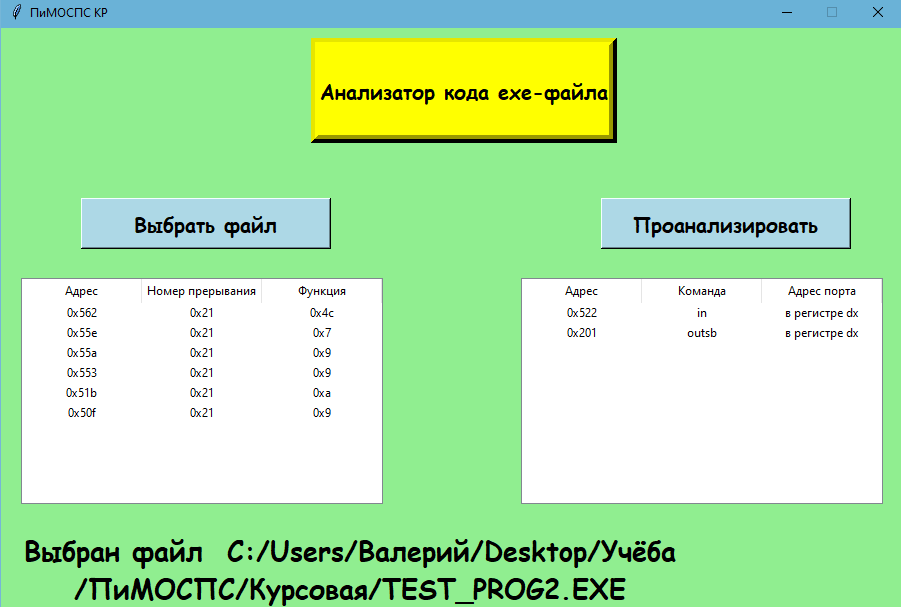


Рис 8. Результат работы программы-анализатора с файлом TEST\_PROG2.exe

## Оценка качества работы алгоритма

Представленный алгоритм способен находить и определять, заданные ему, команды, но не гарантирует достоверность результата. Все «нужные» команды будут найдены, но среди них могут содержаться «лишние» данные, которые на самом деле не являются искомыми.

Поиск команды по содержимому одного байта (команды ввода/вывода) является менее надёжным, т.к. высока вероятность нахождения не тех данных. Проверка двух байт (команда int и, предшествующая ей, команда mov) повышает качество анализа, и обеспечивает меньший процент «мусора».

Плюсом данного алгоритма является быстрота поиска необходимых данных, т.к. производится небольшое число проверок.

# Вывод

В ходе работы был разработан и реализован алгоритм анализа содержимого exe файла. Были выявлены его достоинства и недостатки, так же был предложен вариант улучшения алгоритма. Навыки и знания, приобретенные в ходе написния работы, являются важными и полезными для понимания принципов и методов организации системных программных средств.

# Список источников информации

* <https://frolov-lib.ru/books/bsp.old/v18/ch4.html>
* <http://dfe.petrsu.ru/koi/posob/microcpu/inout.html>
* <https://prog-cpp.ru/micro-interrupts/>
* <http://www.codenet.ru/progr/dos/>
* <http://www.club155.ru/x86cmd>

# Приложения

## Приложение 1 – исходный код программы TEST\_PROG1

;Эта программа воспроизводит звук

;через динамик компьютера

;61h - адрес порта динамика

;42h - адрес порта таймера

d1 segment page public 'data'

Delay DW 3000 ;Длительность звука

d1 ends

st1 segment page stack 'stack'

dw 100 dup (?)

st1 ends

c1 segment page public 'code'

assume cs:c1, ds:d1, ss:st1

start:

IN AL, 61h ;Получить состояние динамика

PUSH AX ;и сохранить его

OR AL, 00000011b ;Установить два младших бита

OUT 61h, AL ;Включить динамик

MOV AL, 10 ;Высота звука (частота)

OUT 42h, AL ;Включить таймер, который

;будет выдавать импульсы на

;динамик с заданной частотой

MOV CX, Delay ;Установить длительность звука

;Цикл, который определяет продолжительность звучания

;Поскольку одного цикла для задержки будет недостаточно

;(пробежит очень быстро и динамик просто не успеет включиться)

;используется вложенный цикл

Zvuk:

PUSH CX

MOV CX, Delay

Cicle:

LOOP Cicle

POP CX

LOOP Zvuk

POP AX ;Получить исходное состояние

AND AL, 11111100b ;Сбросить два младших бита

OUT 61h, AL ;Выключить динамик

mov ah, 7h

int 21h

mov ah, 4ch

int 21h

c1 ends

END start

## Приложение 2 – исходный код программы TEST\_PROG2

; Перестановка a(n),a(n-1),a(n-2),...,a(n/2),a(1),a(2),...,a(n/2-1).

; Многосегментная программа. Создаётся exe файл

d1 segment page public 'data'

mess1 db 'Input: ',10,13,'$'

in\_str db 22 dup (?)

d1 ends

e1 segment page public 'data'

mess2 db 10,13, 'Output: ',10,13,'$'

out\_str db 20 dup ('$')

e1 ends

st1 segment page stack 'stack'

dw 100 dup (?)

st1 ends

c1 segment page public 'code'

assume cs:c1, ds:d1, es:e1, ss:st1

start:

;Вывод строки запроса

mov dx, offset mess1

mov ah, 9

int 21h

;Считывание исходного массива

mov dx, offset in\_str

mov in\_str, 20

mov ah, 10

int 21h

;Инициализация регистра источника данных

;на конец исходного массива

mov cl, in\_str + 1 ;Количество элементов в массиве

xor ch, ch ;Очистка регистра ch

mov ax, offset in\_str+1 ;Адрес, предшествующий первому элементу

add ax,cx ;Запись адреса последнего элемента

mov si,ax ;Инициализация регистра источника данных

;Инициализация регистра приёмника данных

;на начало выходного массива

mov di, offset out\_str

;Определения числа повторений первого цикла - Переставка второй

;части исходного массива в начало нового в обратном порядке.

;Число повторений - количество элементов делить на 2.

;Деление выполняется сдвигом на 1 в право

shr cx,1

m1:

mov al, [si]

mov es:[di],al

dec si

inc di

loop m1

;Инициализация регистра источника данных

;на начало исходного массива

mov si, offset in\_str+2

;Определения числа повторений второго цикла - Переставка первой

;части исходного массива в конец нового.

;Число повторений - количество элементов делить на 2.

mov cl, in\_str + 1

shr cx,1

m2:

mov al,[si]

mov es:[di],al

inc si

inc di

loop m2

;Вывод строки, предшествующей выходному массиву

mov ax, es

mov ds, ax

mov dx, offset mess2

mov ah, 9

int 21h

;Вывод нового массива

mov dx, offset out\_str

mov ah, 9

int 21h

mov ah, 7

int 21h

mov ah, 4ch

int 21h

c1 ends

end start