**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Организация ЭВМ и систем»**

**Тема: Написание собственного прерывания и работа со строками**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 2384 |  | Валеева А.А. |
| Преподаватель |  | Морозов С.М. |

Санкт-Петербург 2023

# Цель работы.

Изучение работы прерываний, написание своего собственного прерывания и освоение работы со строками в программах на языке Ассемблера процессора Intel X86.

# Основные теоретические положения.

1. ​Краткие сведения о прерываниях.

Прерывание – это процесс вызова процедур для выполнения некоторой задачи, обычно связанной с обслуживанием некоторых устройств (обработка сигнала таймера, нажатия клавиши и т.д.). Когда возникает прерывание, процессор прекращает выполнение текущей программы (если её приоритет ниже) и запоминает в стеке вместе с регистром флагов адрес возврата (CS:IP) – места, с которого будет продолжена прерванная программа. Затем в CS:IP загружается адрес программы обработки прерывания и ей передаётся управление.

Адреса 256 программ обработки прерываний, так называемые векторы прерывания, имеют длину по 4 байта (в первых двух хранится значение IP, во вторых – CS) и хранятся в младших 1024 байтах памяти.

Программа обработки прерывания должна заканчиваться инструкцией IRET (возврат из прерывания), по которой из стека восстанавливается адрес возврата и регистр флагов.

Программа обработки прерывания – это отдельная процедура, имеющая структуру:

SUBR\_INT PROC FAR

PUSH AX ; сохранение изменяемых регистров

<действия по обработке прерывания> MOV AL, 20H

OUT 20H, AL

POP AX ; восстановление регистров IRET

SUBR\_INT ENDP

Две последние строки перед восстановлением регистров необходимы

для разрешения обработки прерываний с более низкими уровнями, чем только что обработанное. Этa дoбaвкa нe нужнa для тex вeктopoв пpepывaний, кoтopыe являютcя pacшиpeниями cущecтвующиx пpepывaний, тaким кaк пpepывaниe 1CH, кoтopoe дoбaвляeт кoд к пpepывaнию вpeмeни cутoк.

Программа, использующая новые программы обработки прерываний, при своём завершении должна восстанавливать оригинальные векторы прерываний. Функция 35 прерывания 21H возвращает текущее значение вектора прерывания, помещая значение сегмента в ES, а смещение в BX. В этом случае программа должна содержать следующие инструкции:

тело

; -- в сегменте данных

KEEP\_IP DW 0 ; для хранения смещения KEEP\_CS DW 0 ; и сегмента прерывания

KEEP\_OLD\_VEC EQU KEEP\_IP ; опционально, для удобства перехода на старое

; -- в начале программы

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

Для задания адреса собственного прерывания с заданным номером в таблицу векторов прерываний используется функция 25H прерывания 21H, которая устанавливает вектор прерывания на указанный адрес нового обработчика.

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, 60H ; номер вектора

INT 21H ; меняем прерывание POP DS

В конце программы восстанавливается старый вектор прерывания:

; 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

При использовании функции 25H прерывания 21H DOS знает, что вы делаете, и гарантирует, что в момент записи прерывания будут заблокированы. Поэтому вызывать команды CLI и STI не нужно. Но они понадобятся в случае ручного изменения вектора прерывания (т.е. без вызова INT 21H), чтобы не допустить возможного возникновения ужасной ситуации, когда сегмент был переопределён, а смещение осталось старым (или наоборот).

Прерывания бывают аппаратные (вызываемые в результате сигналов от оборудования) и программные (вызываемые в коде). В лабораторной работе предлагаются к замене вектора́ следующих прерываний:

* 1CH и 08H – можно рассматривать их как аппаратные прерывания,

генерируемые системным таймером; генерируются автоматически 18,2 раза в сек.;

* 09H – аппаратное прерывание, возникающее в результате нажатия клавиш клавиатуры;
* 60H – пользовательское программное прерывание;
* 16H – программное прерывание для ожидания ввода символа с клавиатуры;
* 21H – программное прерывание для вызова сервисов DOS.

Заменённое тело 9h следует завершать не выходом из прерывания (iret), а переходом к выполнению старого тела 9h (использовать команду jmp dword ptr), иначе обработка сигналов клавиатуры будет нарушена. То же самое касается прерываний 16h и 21h.

1. ​Краткие сведения о командах обработки строк.

Для обработки строковых данных ассемблер имеет пять групп команд обработки строк:

* MOVS — переслать один байт или одно слово из одной области памяти в другую;
* LODS — загрузить из памяти один байт в регистр AL или одно слово в регистр AX;
* STOS — записать содержимое регистра AL или AX в память;
* CMPS — сравнить содержимое двух областей памяти, размером в один байт или в одно слово;
* SCAS — сравнить содержимое регистра AL или AX с содержимым памяти.

Каждая команда имеет модификации, указывающие размер операнда: байт (B), слово (W), двойное слово (D). Например: MOVSB, MOVSW, MOVSD.

Эти команды предполагают, что регистры DI и SI содержат относительные адреса, указывающие на необходимые области памяти (для загрузки можно использовать команду LEA). Регистр SI обычно связан с регистром сегмента данных – DS:SI. Регистр DI всегда связан с регистром дополнительного сегмента – ES:DI. Следовательно, команды MOVS, STOS, CMPS и SCAS требуют инициализации регистра ES (обычно адресом в регистре DS).

Префиксы REP/REPE/REPZ/REPNE/REPNZ позволяют этим командам обрабатывать строки любой длины.

В следующем примере выполняется пересылка 20 байт из STRING1 в STRING2. Пусть оба регистра DS и ES уже инициализированы адресом сегмента данных:

STRING1 DB 20 DUP('\*') STRING2 DB 20 DUP(' ') ...

CLD ;Сброс флага DF для направления слева направо MOV CX,20 ;Счётчик на 20 байт

LEA DI,STRING2 ;Адрес области "куда" LEA SI,STRING1 ;Адрес области "откуда" REP MOVSB ;Переслать данные

1. ​Необходимые вспомогательные функции.
   1. ​Ввод строки:

; -- в сегменте данных

EOFLine EQU '$' ;Определение символьной константы "Конец строки"

STR1HEAD DB 50h,0 ; заголовок строки: можно ввести максимум 50h=80 символов. В следующем байте вместо 0 появится фактическое количество введённых символов

STR1 DB 80 DUP('\*'),0AH, 0DH, EOFLine ; Буфер памяти для введённых символов плюс байты для корректного завершения вывода

; -- в сегменте кода StringRead PROC FAR

mov AH,0ah ; функция ввода строки

push dx ; смещение заголовка строки...

int 21h ; вызов функции DOS ввода строки pop bp ; ...поместить в bp

xor bx,bx

mov bl,ds:[bp+1] ; теперь в bx количество введённых символов add bx,bp ; теперь bx указывает на конечный введённый символ

add bx,2 ; теперь bx указывает на байт, следующий за финальным 0dh mov word ptr [bx+1],240ah ;добавить в конец 0ah и '$'

ret

StringRead ENDP

...

mov dx, OFFSET STR1HEAD ; вызов функции ввода строки call StringRead

* 1. ​Задержка во времени:

mov cx,0eh; 14 \* 65535 мкс задержки mov dx,0ffffh; ещё 65535 мкс задержки mov ah,86h; функция "ждать"

int 15h; вызов функции ожидания

# Задание

Состоит из двух основных задач:

* + 1. Реализация сценария работы с прерываниями в соответствии с вариантом.
    2. Реализация преобразования строки с использованием команд работы со строками в соответствии с вариантом.

# Вариант 4. Шифр 4г.

* Действия основной программы: вывод приветственного сообщения и ввод строки, замена 9h, ожидание ввода строки.
* Действия 9h: выполнить работу и восстановить себя

# Замечания:

* 1. В сценариях опущены:

– Вывод приветственного сообщения и ввод строки пользователем. Это 1-ый пункт основной программы каждого сценария

(поэтому основная программа во всех сценариях начинается с пункта 2). Слова «Ожидание ввода строки», которые встречаются в таблице сценариев, – это всего лишь организация задержки с ожиданием нажатия клавиш, для ввода данных не используется.

– Действия по восстановлению изменённых прерываний, если они не требуются по сценарию специально. После завершения программы все изменённые прерывания всегда должны быть восстановлены (независимо от сценария).

* 1. «вып. работу» означает, что надо на основе введённой строки (п. 1 основной программы) создать модифицированную строку и вывести её на экран. При выполнении преобразования нельзя портить исходную строку, результат преобразования должен записываться в выходную строку.
  2. Перед запуском ожидания нажатия клавиши («ввод строки» в таблице) вывести сообщение об этом.
  3. Перед заменой 9h следует сделать небольшую задержку (см. 3.2

«Задержка во времени»), чтобы предшествующая активность пользователя была обработана до того, как 9h будет изменён;

* 1. Для исключения возможного взаимного влияния системных и пользовательских прерываний рекомендуется отвести в программе под стек не менее 1Кбайт.
* Действия по обработке строки: Преобразование введённых во входной строке строчных латинских букв в русские в соответствии с правилами транслитерации.

# Замечания:

1. При выполнении преобразования обязательно использовать команды работы со строками.
2. Завершающие символы (0ah, 0dh, «$») рассматривать именно как завершающие, т.е. не подвергать преобразованиям; закончить ими выходную строку.

# Выполнение работы.

1. Для начала под Stack было выделим 1 кб. Далее в сегменте данных были созданы следующие поля:
   * *InputString* - для начала выделим буфер под вводимую строку размером 100 символов (64h)
   * *header* — это заголовок для этой вводимой строки для обработки, первый байт показывает максимальное количество элементов, а второй реально считанное количество.
   * *ProcessedStr* - далее идёт буфер для строки, которую мы будем преобразовывать, но так как мои числа будут преобразованы в 2 с/сч, то их длина увеличиться, следовательно, увеличиться размер выводимой строки, по сравнению с размером вводимой строки.
   * В переменной *HelloMessage* хранится приветственное сообщение. В переменной *Waiting* хранится сообщение о ожидании ввода.
   * Переменные *KEEP\_IP и KEEP\_CS* отвечают за хранение сегмента и смещения оригинального прерывания 9h.
   * *overwritten* — это переменная нужна как флаг, который показывает, перезаписан ли 9h.
2. Далее идет дополнительные функции — WriteMsg: вывод приветственного сообщения, ReadString: ввод пользовательской строки, RestoreInterruption: восстановление прерывания, ChangeInterruption: замены прерывания, MyFunction: функция обработки строки с прерыванием, а далее основная функция — Main.
3. Вывод строки осуществляется при помощи вызова функции WriteMsg? предварительно требуется поместить адрес начала строки в dx. Внутри функции старший бит выставляется в 9, после чего вызывается прерывание 21.
4. Ввод строки осуществляется при помощи функции ReadString, предварительно требуется поместить адрес начала заголовка строки в dx. Адрес заголовка записывается в bp, после чего вызывается команда 0ah прерывания 21h, которая считывает строку, после чего записывает во второй бит реальное

количество считанных символов. Далее добавляем в bx bp — теперб bx указывает на конечный введенный символ, добавляем 2 - bx указывает на байт, следующий за финальным 0dh. И добавляем в конец строки 0ah и '$'(перевод строки и конец строки).

1. ChangeInterruption — функция замены прерывания. Процедура начинается с сохранения текущего состояния регистров ax, bx и es в стеке. Инструкция cmp проверяет, равна ли переменная overwritten 1. Если это так, программа переходит к функции already\_changed, пропуская остальную часть процедуры. Если флаг не установлен в 1, программа переходит к сохранению исходного вектора прерывания для прерывания 9h. Он использует прерывание DOS 21h с функцией 35h для получения сегмента и смещения вектора прерывания. Они хранятся в KEEP\_CS и KEEP\_IP для последующего использования. Затем программа сохраняет адрес текущего сегмента данных (ds) в стеке. Затем он устанавливает адрес нового обработчика прерываний (myFunction) в dx и ax. Программа использует прерывание DOS 21h с функцией 25h, чтобы задать новый вектор прерывания для прерывания 9h. Меняем переменную flag.
2. RestoreInterruption — функция восстановления прерывания, проверяет переменную overwritten. Если overwritten= 0, то есть прерывание восстановлено, то тогда переходим в функцию already\_restored(там восстанавливаем регистры), пропуская остальную часть процедуры. Иначе восстанавливаем прерывание и меняем флаг.
3. MyFunction — функция с обработкой строки. Мне нужно заменить числа, переведя их 2 СС и разделив пробелом. Начинаем с сохранения текущего состояния регистров. Сохраняем адрес вводимой и обрабатываемой строки (в si и di. Далее функцией strproc мы побайтово считываем нашу строку *InputString,* проверяя символ конца строки, если встретили его, то переходим в процедуру конца процедуры. С помощью stosb записываем содержимое из al в память. Далее каждый символ проверяем на число с помощью кода ASCII, подробнее описано в комментариях к коду, если не число, то сохраняем в строк, если

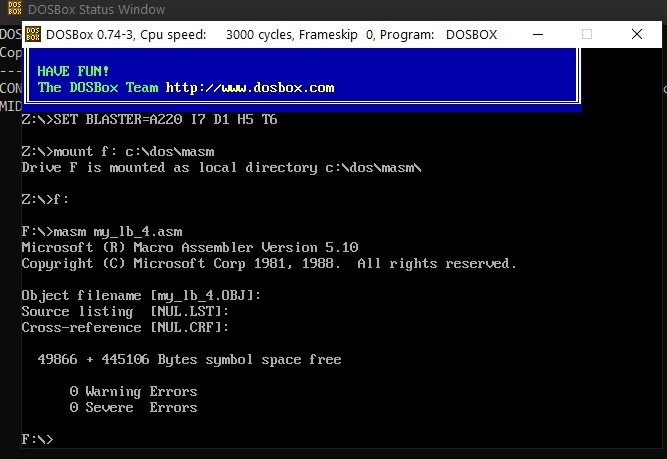
число, то переходим к процедуре обработки. Мы точно знаем длину — 4 символ, так как 9 = 1001. Создаем маску, равную 1000, с помощью нее мы будет сохранять в строку двоичный вид числа, не забывая уменьшая маску побитовым сдвигом. Теперь подробнее о логике обработки: мы побитово через and сравниваем встретившееся число и маску. Например 7 = 111. 111&1000 = 0, так как первый из 4-х символов это 0. Далее маска = 100, 111&100 = 1 и т. д. Если это 1, то записываем ее в строку, с 0 дела обстоят сложнее, так как это может быть незначащий 0. Поэтому через проверку маски мы можем убрать (dec di) записанный 0, если он не нужен. Далее смещаем маску и продолжаем процедуру, пока маска не станет = 0. Записываем число, ставим пробел, увеличиваем бит в di. Восстанавливаем регистры и вызываем оригинальное прерывание 9h.

1. Головная процедура — печатаем строку-приветствие, считываем строку пользователя, печатаем ее для наглядности, делаем задержку 9h, меняем его, печатаем строку-ожидание, ждем нажатие пользователя, восстанавливаем прерывание и возвращаемся из процедуры.

# Тестирование.

Табл. 1

|  |  |  |
| --- | --- | --- |
| Входные данные | Выходные данные | Пояснение |
| 7 | 1 | Строка длиной в один символ, которая должна преобразоваться |
| al9i7na | al 1001 i 111 na | Числа стоят через буквы, тестирование проходит |
| g8g8g88g8gg | G 1000 g 1000 g 1000 1000 g  1000 gg | Цифры, стоящие подряд, разделяются пробелом и корректно преобразуются |





# Вывод

По итогам лабораторной работы были изучены принципы работы прерывания, написано собственное прерывания, а также освоена работа со строками на языке Ассемблера процессора Intel X86.

**ПРИЛОЖЕНИЕ А**

Название файла: my\_lb\_4.asm

AStack SEGMENT STACK

DW 512 DUP(?) ; allocate 1 kb for stack

AStack ENDS

DATA SEGMENT

header DB 64h, 0 ; header for input string

InputString DB 64h DUP('?'), 0dh, 0ah, '$' ; память для вводимой строчки

ProcessedStr DB 190h DUP('?'), 0dh, 0ah, '$' ; буфер для обрабатываемой строки (размер больше,т к больше символов)

HelloMessage DB 'Hello! Input your string: ', 0dh, 0ah, '$'

Waiting DB 'Press any ke...', 0dh, 0ah, '$'

overwritten DB 0 ; flag which shows if 9h is overwritten

KEEP\_IP DW 0 ;хранит смещение прерывания

KEEP\_CS DW 0 ;хранит сегмент заменяемого прерывания (9h)

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, SS:AStack

; функция печати строки

WriteMsg proc near

push ax

mov ah,9

int 21h

pop ax

ret

WriteMsg endp

ReadString PROC ; функция чтения введенной строки

; сохраним регистры

push ax

push bp

push bx

mov ah, 0ah ; ;функция ввода строки

push DX ; смещение заголовка строки

int 21h

pop bp ; помещаем заголовок в bp

xor bx, bx ; обнуляем bx

mov bl, ds:[bp + 1] ; теперь в bl факт.размер введенных символов

add bx, bp ; bx указывает на конечный введенный символ

add bx, 2 ; bx указывает на байт, следующий за финальным 0dh

mov word ptr [bx + 1], 240ah ;обавляем в конец строки 0ah и '$'(перевод строки и конец строки)

;восстановим регистры

pop bx

pop bp

pop ax

ret

ReadString ENDP

; ф-я восстановления прерывания

RestoreInterruption PROC

push ax

push ds

push dx

cmp overwritten, 0 ; если уже восставновили прерывание - то делать нечего

jz already\_restored

; восстановление оригинального 9h

push ds ; сохраняем адрес сегмента данных в стэке

mov dx, KEEP\_IP ; записываем в dx смещение оригинального прерывания 9h

mov ax, KEEP\_CS ; записываем в ax сегмент оригинального прерывания 9h

mov ds, ax ; записываем в ds ax (сегмент оригинального прерывания 9h)

mov ah, 25h ; функция для замены вектора

mov al, 9h ; номер вектора прерывания

int 21h ; восстанавливаем 9h

pop ds ; восстанавливаем сегмент данных

mov overwritten, 0 ; прерывание восстановили

already\_restored:

; восставновление регистров

pop dx

pop ds

pop ax

ret

RestoreInterruption ENDP

ChangeInterruption PROC

push ax

push bx

push es

cmp overwritten, 1 ; если прерывание уже перезаписано, то делать нечего

je already\_changed ; прыгаем в нужную функцию

; запоминание СЕГМЕНТ:СМЕЩЕНИЕ вектора прерывания с номером 9h

mov ah, 35h ; функция для получения СЕГМЕНТ:СМЕЩЕНИЕ вектора прерывания

mov al, 9h ; номер вектора прерывания

int 21h ; прерывание записывает сегмент прерывания в es, а смещение в bx

mov KEEP\_IP, bx ; запоминаем смещение в KEEP\_IP

mov KEEP\_CS, es ; запоминаем сегмент в KEEP\_ES

; замена прерывания 9h на пользовательское

push ds ; сохраняем адрес сегмента данных в стэке

mov dx, offset MyFunction ; записываем смещение моей функции в dx

mov ax, seg MyFunction ; записываем сегмент моей функции в ax

mov ds, ax ; записываем в ds значение ax

mov ah, 25h ; функция для замены вектора

mov al, 9h ; номер вектора для замены

int 21h ; заменяем 9h

pop ds ; восстанавливаем сегмент данных

mov overwritten, 1; теперь прерывание перезаписано

already\_changed:

pop es

pop bx

pop ax

ret

ChangeInterruption ENDP

MyFunction PROC ;(исправлено для 3 задания на зищиту)

;восстановление регистров

push ax

push si

push di

push dx

mov ax, DATA ; записываем сегмент данных в ax

mov ds, ax ; инициализируем ds

mov es, ax ; и es

cld ; сброс флага для продвижения вперед

lea si, InputString ; в si сохраняем адрес моей входной строки

lea di, ProcessedStr ; а в di - выводимой (обрабатываемой)

strproc:

lodsb ; побайтово загружаем входную строку для обработки

cmp al, '$' ;сравниваем с символом конца строки

je end\_proc ; если он - то переходим в конец

cmp al, '0' ; если текущий символ по ASCII меньше 0

jl notDigit

cmp al, 'F' ;или же больше F -> это не шестнадцатеричное число

jg notDigit ; ничего с ним не делаем, просто пишем в новую строку

; иначе - это шестнадцатеричное число

mov dl, al ; скопируем оригинал числа

sub dl, '0' ; через ASCII делаем число-строку числом

mov ah, 1000b ; идея преобразования - через маску 8=1000(2)

; преобразование шестнадцатеричного символа в двоичное представление

cmp dl, 9

jle hex\_digit

sub dl, 7 ; для символов A-F вычитаем 7

hex\_digit:

mov bx, offset ProcessedStr; копируем смещение обрабатываемой строки

cmp di, bx ; сравниваем текущее смещение

je skip

mov al, ' ' ; пробел перед числом

stosb; запишем содержимое(пробел) из al в память(пробел)

skip:

mov cl, 4

mov bh, 1

push di

convert:

mov dh, ah; скопируем маску в dh

and dh, dl ; тут происходит побитовое сравнение маски с числом (в2) &

cmp dh, 0 ; сравним полученный результат (он записывается в dh) с 0

jnz one; если не 0

zero: ; обработка случая с 0

mov al, '0' ; в al копируем 0

stosb; запишем содержимое из al в память(пробел)

cmp bh, 0

je next ;число 0 - пишем 0

dec di ; если нет, то это 0 в начале числа, его убираем

jmp next; переход к следующему

one:

mov al, '1' ; тогда это 1, которую сохр в нашу строку

stosb ; запишем содержимое из al в память

mov bh, 0

next: ; переход дальше

shr ah, 1; через сдвиг право делим маску на 2(1000->100->10..)

loop convert; дальше побитово сравниваем (пока не 0)

convert\_end: ; маска 0, дошли до конца числа

pop dx

cmp di, dx ;

jnz add\_space; если не равно, то пробел

inc di ; +1

add\_space:

; когда маска 0, не забываем отделить полученное число пробелом

mov al, ' '

;ф-я, если не число

notDigit:

stosb ; просто берем символ из al

jmp strproc ; и продолжаем читать строку

end\_proc: ; конец чтения строки

stosb; запишем содержимое из al в память

mov dx, offset ProcessedStr ; сохраняем смещение ProcessedStr в dx

call WriteMsg ; печать строки

mov al, 20h ; для разрешения обработки прерываний

out 20h, al ; более низким уровнем, чем обработанное

pop dx

pop di

pop si

pop ax

call RestoreInterruption ; ф-я восстановления прерывания

jmp dword ptr [KEEP\_IP] ; вызываем оригинальное прерывание 9h

MyFunction ENDP

; Головная процедура

Main PROC FAR

push ds ;сохраняем в стэк адрес начала PSP

sub ax, ax ; обнуляем регистр ax

push ax ; сохраним нулевое смещение

mov ax, DATA ; переносим в решистра сегмент data

mov ds, ax ; иниц-я ds

mov es, ax ; и es

; тут печатаем строку

mov dx, offset HelloMessage ; копирование смещения HelloMessage в dx

call WriteMsg ; вызываем ф-ю печати

; теперь считываем строку пользователя

mov dx, offset header ; копируем заголовок смешения

call ReadString

; задержка между вводом и выводом (2 задание на защиту)

mov cx, 0013h ; Параметры для задержки в 1.25 секунды (=19 в 10 с/сч)

mov dx, 12D0h

mov ah, 86h

int 15h

;выведем на экран строчку, введенную пользователем

mov dx, offset InputString

call WriteMsg

; задержка по времени перед заменой 9h

mov cx, 0eh ; 14 \* 65535 мкс задержки

mov dx, 0ffffh ; ещё 65535 мкс задержки

mov ah, 86h ; функция "ждать"

int 15h ; вызов функции ожидания

;через ф-ю для замены прерываний меняем 9h

call ChangeInterruption

;печатаем строку-ожидания нажатия

mov dx, offset Waiting

call WriteMsg

; ждем ввод

mov ah, 0 ; 0 - это команда для считывания одного ключа (al будет иметь ascii-код, а ah - флаги)

int 16h ; ожидание нажатия кнопки

;done:

; восстанавляем прерывание в любом случае

ret

Main ENDP

CODE ENDS

END Main