**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

## ОТЧЕТ

**по лабораторной работе №4**

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

**Тема: «Написание собственного прерывания**

**и работа со строками.»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Титкова С.Д. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2024

## Цель работы.

Изучить особенности прерываний на языке Ассемблер, написать собственное прерывание, ознакомиться с работой со строками и реализовать программу, охватывающую темы прерываний и обработки строк.

## Основные теоретические положения.

**1.** 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) – ме́ста, 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.

В лабораторной работе среди действий по обработке прерывания могут быть:

– преобразование строки;

– вывод сообщения на экран;

– замена или восстановление вектора прерывания (другого или самого́ себя);

– вызов прерывания;

– и др.

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:

; -- 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 тело

; -- 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.

**2. 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

## Задание.

Вариант работы 22.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6 | 2) замена 1ch; 3) ожидание ввода строки; 4) вызов 4bh. | после замены через 3 сек. заменяет 4bh и восст. себя |  | 1) вып. работу;  2) восст. себя |

ф) Исключение русских букв (включая "ё", "Ё"), введённых во входной строке, при формировании выходной строки.

## Ход выполнения работы.

1. В начале программы определяются сегменты стека и данных. Сегмент стека выделяет память для хранения временных данных, таких как адреса возврата и локальные переменные. Размер стека установлен в 1024 байта (400h). Сегмент данных содержит переменные для хранения сообщений, буферов для ввода и вывода, а также переменные для хранения адресов старых обработчиков прерываний.

Далее происходит сохранение адреса старого обработчика прерывания 1Ch. Это позволяет программе запомнить старый адрес обработчика для последующего восстановления.

После сохранения старых адресов устанавливается новый обработчик прерывания. Этот новый обработчик будет вызывать функцию обработки строки.

После установки нового обработчика программа ожидает ввода от пользователя. В основном коде происходит вывод приветственного сообщения и чтение строки от пользователя. После нажатия клавиши вызывается пользовательское прерывание 4Bh для обработки введенной строки.

После выполнения всех операций происходит восстановление старых обработчиков прерываний и завершение программы.

2. В обработчике прерывания 4Bh происходит обработка строки. В начале функции сохраняются регистры SI, DI, BX, CX, AX и ES на стеке. Это необходимо для восстановления их значений после завершения работы функции.

Сегмент данных загружается в регистр DS, а сегмент для выходного буфера — в ES. Это позволяет программе правильно обращаться к переменным в этих сегментах.

Указатель DI устанавливается на начало выходного буфера (output), а указатель SI — на начало входного буфера (input).

Регистр CX обнуляется и затем заполняется размером введенной строки (изначально указанным в заголовке) с добавлением трех дополнительных байтов для корректного завершения строки.

Функция использует цикл для обработки каждого символа входной строки:

Команда lodsb загружает текущий байт из входного буфера в регистр AL и увеличивает указатель SI.

Затем происходит проверка загруженного символа на принадлежность к диапазону нежелательных (русских) букв.

Если символ не является нежелательным, он записывается в выходной буфер с помощью команды stosb.

Если загруженный символ является нежелательным (русская буква, включая ё), он пропускается.

После завершения цикла выводится преобразованная строка с помощью вызова процедуры writeMsg.

В конце функции восстанавливаются все ранее сохраненные регистры из стека. Завершается выполнение функции командой ret, возвращая управление обратно к месту вызова.

3. Тестирование прерывания и обработки строки.

В начале пользователь вводит строку и нажимает enter (рисунок 1).



Рисунок 1- приветственное сообщение и ввод строки

Далее возможны 2 сценария:

3.1. Ждём 3 секунды, как указано в задании, при этом можно нажимать любые клавиши, кроме enter. Ничего не будет происходить. Когда ещё раз будет нажат enter, выведится обработанная строка и программа завершится. (Рисунок 2)

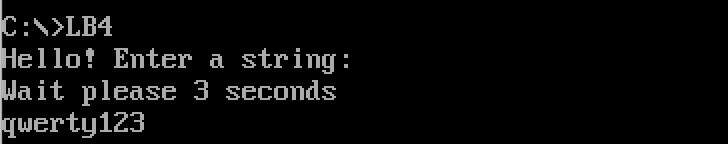


Рисунок 2 – ожидание 3 секунд и вывод обработанной строки

3.2. Если не дождались 3 секунд, то при нажатии второй раз на enter, то ничего не выведится и программа завершится. (Рисунок 3)

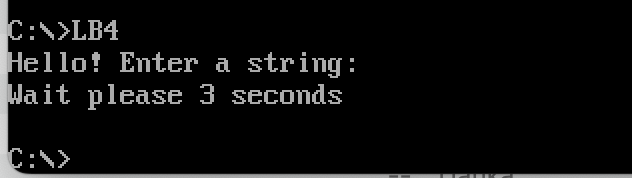


Рисунок 3 – нажали повторно enter, не дождавшись 3 секунд

# 4. Тестирование различных входных данных

|  |  |  |  |
| --- | --- | --- | --- |
| № | Ввод | Вывод | Ожидаемый вывод |
| 1 | ЁHello, world! | Hello, world! | Hello, world! |
| 2 | abcёfgrЁ34 | abcfgr34 | abcfgr34 |
| 3 | Fфhеhklk12ва | Fhhklk12 | Fhhklk12 |

## Вывод.

В ходе выполнения лабораторной работы были изучены особенности прерываний на языке Ассемблер, написаны собственные прерывания. Также в результате изучения работы со строками была реализована программа, демонстрирующая преобразование строки с использованием собственных прерываний.

**Приложение.**

Код программы LB4.asm:

.model small

.stack 400h

.data

keep\_ip\_1ch DW 0 ; для хранения смещения

keep\_cs\_1ch DW 0 ; для хранения сегмента

keep\_ip\_4bh DW 0 ; для хранения смещения

keep\_cs\_4bh DW 0 ; для хранения сегмента

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

message db "Hello! Enter a string: ", 0ah, 0dh, EOFLine

change db "Wait please 3 seconds", 0ah, 0dh, EOFLine

input\_header DB 64H, 0 ; заголовок строки: можно ввести максимум 64h=100 символов.

; В след байте вместо 0 появится фактическое количество введённых символов

input DB 64H DUP('\*'), 0ah, 0dh, EOFLine ; Буфер памяти для введённых символов

; плюс байты для корректного завершения вывода

output DB 64H DUP('\*'), 0ah, 0dh, EOFLine ; Буфер памяти для обработанных символов

; плюс байты для корректного завершения вывода

count\_1ch dw 0 ; счетчик вызовов 1ch

.code

mov ax, @data

mov ds, ax ; соответствие регистру ds - сегмента данных

; вывод приветствия

mov dx, OFFSET message

call writeMsg

; ввод строки пользователем

mov dx, OFFSET input\_header

call stringRead

; сохранение позиций старого обработчика прерывания 4bh

mov ah, 35h ; функция получения вектора

mov al, 4bh ; номер вектора

int 21h

mov keep\_cs\_4bh, es ; запоминание сегмента оригинального обработчика

mov keep\_ip\_4bh, bx ; запоминание смещения оригинального обработчика

; сохранение позиций старого обработчика прерывания 1ch

mov ah, 35h ; функция получения вектора

mov al, 1ch ; номер вектора

int 21h

mov keep\_cs\_1ch, es ; запоминание сегмента оригинального обработчика

mov keep\_ip\_1ch, bx ; запоминание смещения оригинального обработчика

; 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

; замена обработчика прерывания 1ch

push ds

mov dx, offset proccesor\_1ch ; смещение для процедуры в dx proccesor\_1h - новый обработчик прерывания 1h

mov ax, seg proccesor\_1ch ; сегмент процедуры в ax

mov ds, ax ; помещаем в DS

mov ah, 25h ; функция установки вектора прерывания

mov al, 1ch ; номер вектора

int 21h ; замена прерывания

pop ds

; сообщение об ожидании

lea dx, change

call writeMsg

waiting:

mov ah, 0h ; функция считывания нажатых клавишь

int 16h ; вызов функции 0 прерывания int 16

cmp ah, 1Ch ; проверка скан-кода считанной клавиши на соответствие скан-коду enter

jne waiting ; переход на метку waiting, если скан-коды не совпадают

int 4bh

; восстановление 1ch

push ds

mov dx, keep\_ip\_1ch ; смещение для оригинального обработчика 1ch

mov ax, keep\_cs\_1ch ; сегмент оригинального обработчика

mov ds, ax ; помещаем в DS

mov ah, 25h ; функция установки вектора прерывания

mov al, 1ch ; номер вектора

int 21h ; замена прерывания

pop ds

; восстановление 4bh

push ds

mov dx, keep\_ip\_4bh ; смещение для оригинального обработчика 4Bh

mov ax, keep\_cs\_4bh ; сегмент оригинального обработчика

mov ds, ax ; помещаем в DS

mov ah, 25h ; функция установки вектора прерывания

mov al, 4bh ; номер вектора

int 21h ; замена прерывания

pop ds

; завершение программы

mov ah, 4ch

int 21h

StringRead PROC

; запоминание изменеяемых регистров

push ax

push bp

push bx

mov ah, 0ah ; функция ввода строка

push dx ; смещение заголовка строки

int 21h ; вызов функции DOS ввода строки

pop bp ; поместить в bp

xor bx, bx ; обнуление bx

mov BL, ds:[bp + 1] ; теперь в bx количество введенных 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 '$'

; возвращение сохраненных регистров

pop bx

pop bp

pop ax

RET

StringRead ENDP

writeMsg proc

push ax

mov ah, 9h

int 21h

pop ax

ret

writeMsg ENDP

convertSTR proc

; запоминание изменеяемых регистров

push si

push di

push bx

push cx

push ax

push es

; загрузка сегмента данных в регистр dx

mov ax, @data

mov ds, ax

mov ES, AX

cld ; Сброс флага DF для направления слева направо

lea di, output ; Адрес области "куда"

lea si, input ; Адрес области "откуда"

xor cx, cx ; обнуление регистра cx

mov cl, input\_header[1] ; сl - размер строки

add cl, 3 ; + дополнительные символы

;сравнение символов

convert\_loop:

lodsb ;читает из входной строки(input), регистр si увеличивается на 1

cmp al, 80h ;А

jb write ;если меньше,то записываем

cmp al, 0afh ;п

ja proccess ;если больше,то идём на следующую проверку

jmp skip\_symbol ;если меньше,то пропускаем

; продолжение проверки

proccess:

cmp al, 0e0h ;р

jb write ;если меньше,то записываем

cmp al, 0F1h ;ё

jbe skip\_symbol;если меньше или равен, то пропускаем

;записываем символы

write:

stosb ;записывает в output, регистр di увеличивается на 1

;пропускаем сим

skip\_symbol:

loop convert\_loop

; вывод преобразованной строки

mov dx, offset output

call writeMsg

; возвращение сохраненных регистров

pop es

pop ax

pop cx

pop bx

pop di

pop si

ret

convertSTR endp

; обработчик прерывания 1Ch

proccesor\_1ch proc

inc count\_1ch ; увеличение количество запусков

cmp count\_1ch, 54 ; если не 54 запуск, то переход к конце

jne skip

; замена обработчика прерывания 4bh

push ds

mov dx, offset proccesor\_4bh ; смещение для процедуры в dx proccesor\_4Bh - новый обработчик прерывания 4Bh

mov ax, seg proccesor\_4bh ; сегмент процедуры в ax

mov ds, ax ; помещаем в DS

mov ah, 25h ; функция установки вектора прерывания

mov al, 4bh ; номер вектора

int 21h ; замена прерывания

pop ds

; восстановление 1ch

push ds

mov dx, keep\_ip\_1ch ; смещение для оригинального обработчика 1ch

mov ax, keep\_cs\_1ch ; сегмент оригинального обработчика

mov ds, ax ; помещаем в DS

mov ah, 25h ; функция установки вектора прерывания

mov al, 1ch ; номер вектора

int 21h ; замена прерывания

pop ds

skip:

iret

proccesor\_1ch endp

; обработчик прерывания 4Bh

proccesor\_4bh proc

; сохранение изменяемых регистров

push ax

push dx

call convertSTR

; восстановление 4bh

push ds

mov dx, keep\_ip\_4bh ; смещение для оригинального обработчика 4Bh

mov ax, keep\_cs\_4bh ; сегмент оригинального обработчика

mov ds, ax ; помещаем в DS

mov ah, 25h ; функция установки вектора прерывания

mov al, 4bh ; номер вектора

int 21h ; замена прерывания

pop ds

pop dx

pop ax

iret

proccesor\_4bh endp

end