# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

# «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

**Кафедра МО ЭВМ**

# ОТЧЕТ

**по лабораторной работе №4**

# по дисциплине «Организация систем и ЭВМ»

**Тема:**

«**Написание собственного прерывания и работа со строками**»

Студент гр. 3388 Дубровин Д.Н.

Преподаватель Фирсов М.А.

Санкт-Петербург 2024

# Цель работы.

Изучить работу с прерываниями и обработку строк в ассемблере 8086. Написать программу, реализующую собственное прерывание 9h с обработкой строки.

# 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 7: 7ё

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | 2) замена 9h;  3) замена 1ch;  4) ожидание ввода строки. | после замены через 6 сек. восст. 9h и себя. | вып. работу |  |

ё) Инвертирование введённых во входной строке цифр в десятичной СС.

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). Слова́ «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). Перед завершением программы следует восстановить в том числе и те вектора прерываний, которые могли бы восстановиться по таймеру. После завершения программы DOS освобождает память, занимаемую программой, что может помешать корректному восстановлению векторов. Поэтому рассчитывать на восстановление по таймеру после завершения программы будет неправильно.

2) «Ожидание ввода строки» следует делать так, чтобы пользователь мог нажать множество клавиш, прежде чем решит завершить ввод нажатием Enter.

4) «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.

5) Перед запуском ожидания нажатия клавиши («ожидание ввода строки» в таблице) вывести сообщение об этом.

6) 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;

7) Для исключения возможного взаимного влияния системных и пользовательских прерываний рекомендуется отвести в программе под стек не менее 1Кбайт.

# Основные теоретические положения.

1. Краткие сведения о прерываниях.

Прерывание – это процесс вызова процедур для выполнения некоторой задачи, обычно связанной с обслуживанием некоторых устройств. 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.

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);

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.

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.

При использовании функции 25H прерывания 21H DOS знает, что вы делаете, и гарантирует, что в момент записи прерывания будут заблокированы.

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

– 1CH – можно рассматривать как аппаратное прерывание, генерируемое системным таймером; генерируются автоматически 18,2 раза в сек.;

– 09H – аппаратное прерывание, возникающее в результате нажатия клавиш клавиатуры;

– 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;

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.

# Выполнение работы

# В главной процедуре main происходит сохранение прежних обработчиков прерываний 9h и 1ch, вывод сообщения-приветствия и считывание строки. Затем заменяется обработчик 9h в котором выполняется процедура revert\_digit, ответственная за преобразование введённой строки и вывод изменённой строки. Затем происходит замена обработчика прерывания 1ch, в котором восстанавливаются оригинальные векторы прерываний 1ch и 9h с выводом соответствующих сообщений в том случае, если прошло 6 секунд с момента вызова. В течение 6 секунд в ответ на любую нажатую клавишу кроме enter программа будет дважды выводить измененную строку. Через 6 секунд регистры восстановятся и для завершения программы пользователь должен нажать enter. Если пользователь не дождался 6 секунд и нажал enter, то программа восстановит оригинальные векторы прерываний в процедуре main и выведет соответствующие сообщения.

# В процедуре revert\_digits происходит обработка строки, введенной пользователем. В регистры ES и DS сохраняется адрес сегмента данных, это необходимо, так как регистры SI и DI, необходимые для использования MOVSB, связаны с ними.

# В процедуре revert\_digits происходит посимвольное преобразование исходной строки, если символ является числом, то оно инвертируется в 10-ричной системе счисления по формуле (105 - x) и добавляется в выходную строку.

# Если символ не является числом, то он заноситься в выходную строку без изменения.

# Тестирование прерываний.

# Прерывание 9H реагирует на нажатие клавиш, поэтому после ввода строки пользователь должен нажать любую клавишу для вывода изменённой строки. Измененные строки будут выводиться в течение 6 секунд, после чего произойдет восстановление прерываний. Для завершения работы пользователь должен нажать клавишу Enter.

# Рисунок 1 — Обработка множественного нажатия клавиш с 6 секундным ожиданием восстановления прерываний

# Рисунок 2 — пользователь ввёл строку и нажал на клавиши, после этого пользователь нажал Enter не подождав 6 секунд. (восстановление прерываний произошло в мейне с соответствующим сообщением)

# Рисунок 3 — программа не завершается пока пользователь не введёт Enter

# Рисунок 4 - пользователь не нажимал клавиш 6 секунд после ввода строки

# 4. Тестирование различных входных данных

|  |  |
| --- | --- |
| Input | Output |
| abcd12345 | abcd87654 |
| 098765 | 901234 |
|  |  |
| 12345abc98765 | 87654abc01234 |

**Вывод**

В ходе лабораторной работы были изучены прерывания и работа со строками на языке Ассемблер Intel 8086. Разработаны собственные обработчики прерываний 9h и 1ch и логика изменения строки.

# ПРИЛОЖЕНИЕ

# ИСХОДНЫЙ КОД ПРОГРАММЫ

1. LR4TEST.ASM

.model small

.stack 400h

.data

keep\_ip\_1ch dw 0 ; для хранения смещения

keep\_cs\_1ch dw 0 ; для хранения сегмента

keep\_ip\_9h dw 0 ; для хранения смещения

keep\_cs\_9h dw 0 ; для хранения сегмента

counter\_1ch dw 0 ; счетчик вызовов 1ch

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

hello\_message db "Здравствуйте! Введите строку: ", 0ah, 0dh, EOFLine

output\_message db "Обработанная строка:", 0ah, 0dh, EOFLine

replace\_1ch db "Обработчик 1ch заменен", 0ah, 0dh, EOFLine

recover\_1ch db "Обработчик 1ch восстановлен", 0ah, 0dh, EOFLine

recover\_main\_1ch db "Обработчик 1ch восстановлен в main", 0ah, 0dh, EOFLine

replace\_9h db "Обработчик 9h заменен", 0ah, 0dh, EOFLine

recover\_9h db "Обработчик 9h восстановлен", 0ah, 0dh, EOFLine

recover\_main\_9h db "Обработчик 9h восстановлен в main", 0ah, 0dh, EOFLine

input\_head 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

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

.code

mov ax, @data

mov ds, ax ; соответствие регистру ds - сегмента данных

; вывод приветствия

mov dx, OFFSET hello\_message

call write\_msg

; ввод строки пользователем

mov dx, OFFSET input\_head

call read\_string

; сохранение позиций старого обработчика прерывания 9h

mov ah, 35h ; функция получения вектора

mov al, 9h ; номер вектора

int 21h

mov keep\_cs\_9h, es ; запоминание сегмента оригинального обработчика

mov keep\_ip\_9h, bx ; запоминание смещения оригинального обработчика

; задержка во времени

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

mov dx, 424fh ; ещё 16975 мкс задержки (суммарно 1с)

mov ah, 86h ; функция "ждать"

int 15h ; вызов функции ожидания

; замена обработчика прерывания 9h

push ds

mov dx, offset handler\_9h ; смещение для процедуры в dx handler\_9h - новый обработчик прерывания 9h

mov ax, seg handler\_9h ; сегмент процедуры в ax

mov ds, ax ; помещаем в DS

mov ah, 25h ; функция установки вектора прерывания

mov al, 9h ; номер вектора

int 21h ; замена прерывания

pop ds

; вывод сообщение о замене прерывания 9h

lea dx, replace\_9h

call write\_msg

; 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

; замена обработчика прерывания 1ch

push ds

mov dx, offset handler\_1ch ; смещение для процедуры в dx handler\_1ch - новый обработчик прерывания 1ch

mov ax, seg handler\_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

; вывод сообщение о замене прерывания 1ch

lea dx, replace\_1ch

call write\_msg

wait\_for\_enter: ; ожидание нажатия Enter, для выхода из программы

mov ah, 0h ; функция "проверка клавиши"

int 16h ; вызываем прерывание для чтения клавиши

cmp ah, 1Ch ; проверяем, что нажата клавиша Enter (код 0x0D)

jne wait\_for\_enter ; если нет, продолжаем ожидание

; 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

; вывод сообщения о восстановлении прерывания 1h

lea dx, recover\_main\_1ch

call write\_msg

mov ax, 0

cmp ax, keep\_cs\_9h ; Восстанавливаем 9h, если есть данные о нём, то восстанавливаем

je finish ; иначе он не заменялся

; восстановление 9h, если было изменено

push ds

mov dx, keep\_ip\_9h ; смещение для оригинального обработчика 9h

mov ax, keep\_cs\_9h ; сегмент оригинального обработчика

mov ds, ax ; помещаем в DS

mov ah, 25h ; функция установки вектора прерывания

mov al, 9h ; номер вектора

int 21h ; замена прерывания

pop ds

; вывод сообщение о восстановлении прерывания 9h

lea dx, recover\_main\_9h

call write\_msg

finish: ; завершение программы

mov ax, 4c00h

int 21h

; обработчик прерывания 1Ch

handler\_1ch proc far

push ax

push dx

push ds

push es

inc counter\_1ch ; увеличение количества запусков

cmp counter\_1ch, 110 ; 18,2 \* 6 = 109,2, если не прошло 110 запусков(6 сек), то переход к концу

jl skip

; Восстановление оригинального обработчика 9h

push ds

mov dx, keep\_ip\_9h ; смещение для оригинального обработчика 9h

mov ax, keep\_cs\_9h ; сегмент оригинального обработчика

mov ds, ax ; помещаем в DS

mov ah, 25h ; функция установки вектора прерывания

mov al, 9h ; номер вектора

int 21h ; замена прерывания

pop ds

; Вывод сообщения о восстановлении прерывания 9h

lea dx, recover\_9h

call write\_msg

; Восстановление оригинального обработчика 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

; Вывод сообщения о восстановлении прерывания 1ch

lea dx, recover\_1ch

call write\_msg

skip:

pop es

pop ds

pop dx

pop ax

iret

handler\_1ch endp

; обработчик прерывания 9h

handler\_9h proc far

; запоминание изменеяемых регистров

push ax

push es

push ds

push dx

lea dx, output\_message ; Выводим строку output\_message

call write\_msg

call invert\_digits ; Вызов функции обработки строки

; возвращение сохраненных регистров

pop dx

pop ds

pop es

pop ax

jmp dword ptr [keep\_ip\_9h] ; Переход к оригинальному вектору

handler\_9h endp

read\_string 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

read\_string endp

write\_msg proc

push ax

mov ah, 9h ; Установление функции вывода строки

int 21h ; Вызов функции вывода строки

pop ax

ret

write\_msg ENDP

invert\_digits proc

; запоминание изменеяемых регистров

push si

push di

push cx

push ax

push es

push ds

; загрузка сегмента данных в регистр dx

mov ax, @data

mov ds, ax

mov es, ax

cld ; Сброс флага DF для направления слева направо

lea si, input ; Адрес области "откуда"

lea di, output ; Адрес области "куда"

mov cx, 0 ; обнуление регистра cx

mov cl, input\_head[1] ; сх - размер строки

add cl, 3 ; + дополнительные символы

invert\_loop:

lodsb ; загрузить из памяти один байт в регистр al

cmp al, '0'

jb add\_output ; если меньше '0', записать в вывод

cmp al, '9'

ja add\_output ; если больше '9', записать в вывод

invert:

neg al ; -x

add al, 105 ; получаем инвертированное число (105 - x)

add\_output:

stosb ; здесь происходит запись числа из регистра al

next:

loop invert\_loop ; переход к следующей итерации

; вывод преобразованной строки

lea dx, output

call write\_msg

; возвращение сохраненных регистров

pop ds

pop es

pop ax

pop cx

pop di

pop si

ret

invert\_digits endp

end