# 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 Сабалиров М.З.

Преподаватель Фирсов М.А.

Санкт-Петербург 2024

# Цель работы.

Изучить работу с прерываниями и обработку строк в ассемблере 8086. Написать программу, реализующую собственное прерывание 4bh 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 18: 2р

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 2) замена 4bh;  3) вызов 4bh;  4) вызов 4bh. |  |  | вып. работу, если уже был вызван; иначе отметить вызов. |

р) Входная строка делится на подстоки символами-разделителями: "/" и "\". В выходную строку вывести все подстроки без разделителей, при этом символы подстрок, перед которыми стоит "/", выводить в прямом порядке, а символы подстрок, перед которыми стоит "\", выводить в инверсном порядке. Например: "абв/гд\1234/\abc/" -> "абвгд4321cba".

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.

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.

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 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;

– 4BH – зарезервированное прерывание, которое при выполнении лабораторной работы можно использовать в качестве пользовательского программного прерывания;

– 16H – программное прерывание для ожидания ввода символа с клавиатуры;

– 21H – программное прерывание для вызова сервисов DOS.

2. Краткие сведения о командах обработки строк.

— LODS — загрузить из памяти один байт в регистр AL или одно слово в регистр AX;

— STOS — записать содержимое регистра AL или AX в память;

Каждая команда имеет модификации, указывающие размер операнда: байт (B), слово (W), двойное слово (D). Например: LODSB, LODSW, LODSD.

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 основной процедуре Main происходит вывод вспомогательных сообщений и ввод строки пользователем.

# Далее с помощью кода, данного в теоретических положениях,

# происходит сохранение оригинального прерывания в переменные KEEP\_IP и KEEP\_CS, также происходит замена прерывания 4bh на пользовательское, реализованное в функции ProcessString и соответствующее условию варианта.

# В обработчике прерывания ProcessString 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 LODSB и STOSB, связаны с ними.

# Также организован условный оператор, гарантирующий, что обработка и вывод результата произойдут только один раз.

# Дальше реализован цикл, проходящий по всей строке и находящий находящий сепаратор и при необходимости подстрока инвертируется

# Тестирование прерывания и обработки строки. Программа просит ввести строку, после ввода строки и нажатия ENTER выводит обработанную строку

# Рисунок 1

# 4. Тестирование различных входных данных

|  |  |
| --- | --- |
| Input | Output |
| ///aaa///bbb | aaabbb |
| /\//\\ |  |
| abc/fg\12345/ | abcfg54321 |
| \1234 | 4321 |
| \1\\23/\/\67 | 13276 |

**Вывод**

В ходе лабораторной работы были изучены прерывания и работа со строками на языке Ассемблер Intel 8086. Разработан собственный обработчик прерывания 4bh и логика изменения строки.

# ПРИЛОЖЕНИЕ

# ИСХОДНЫЙ КОД ПРОГРАММЫ

1. lb4.asm

; Стек программы

AStack SEGMENT STACK

DB 1024 DUP(0) ; 1 КБайт для исключения возможного взаимного влияния системных и пользовательских прерываний

AStack ENDS

; Данные программы

DATA SEGMENT

KEEP\_IP DW 0 ; для хранения смещения

KEEP\_CS DW 0 ; и сегмента прерывания

COMPLETE\_FLAG DB 0 ; флаг проверки того, что прерывание было выполнено

LAST\_SEP\_INDEX DW 0 ; Индекс последнего сепаратора

IS\_SEP\_INVERSE DB 0 ; Флаг, того, надо ли инвертировать предыдщий блок

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

STR2 DB 80 DUP('\*'), 0AH, 0DH, EOFLine ; Буфер памяти для обработанной строки плюс байты для корректного завершения вывода

START\_MESSAGE DB 'Здравствуйте! Введите строку: $' ; Стартовое сообщенте

NEWLINE DB 0Dh, 0Ah, '$' ; Перевод на новую строку

DATA ENDS

; Код программы

CODE SEGMENT

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

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 ; Занулить 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

WriteMsg PROC FAR

mov ah, 9

int 21h ; Вызов функции DOS по прерыванию

ret

WriteMsg ENDP

ProcessString PROC FAR

; Сохраняем все регистры, которые будут изменены

push ax

push bx

push dx

push ds

push es

push si

push di

; сохранение в ds, es адрес начала сегмента даных

mov ax, DATA

mov ds, ax

mov es, ax

; Проверяем, что обработка была только один раз, иначе пропускаем обработку строки

cmp COMPLETE\_FLAG, 1

je END\_PROCESS

; Пользовательская обработка строки (обработчик прерывания)

lea si, STR1 ; Указатель на начало строки STR1

lea di, STR2 ; Указатель на начало строки STR2

ITERATION:

lodsb ; Загружаем в al следующий символ STR1 и увеличиваем si

cmp al, '/' ; Проверка на то, является ли символ сепаратором

je SEPARATOR ; Переход к обработке сепараторов

cmp al, '\' ; Проверка на то, является ли символ сепаратором

je SEPARATOR ; Переход к обработке сепараторов

cmp al, '$' ; Если конец, то дообратываем текущий блок

je SEPARATOR ; Переход к обработке сепараторов

jmp WRITE ; Переход к записи символа в строку

SEPARATOR:

cmp IS\_SEP\_INVERSE, 0 ; Если предыдущий сепаратор не инверсивный, просто обновляем данные о сепараторах

je UPDATE ; Переход к обновлению даннных

mov dx, di ; сохраняем значение di, чтобы потом востановить

mov bx, LAST\_SEP\_INDEX ; Фиксируем адрес предыдущего сепаратора

inc bx ; Добавляем один, чтобы в конце не обработать лишний символ

cmp al, '$'

jne REVERSE

sub si, 2 ; Если конец то не обработываем симвомвыло корректного завершения строки

sub di, 2

REVERSE:

cmp bx, si ; Если дошли до конца, то идем к востановлению di

je LOAD\_DI

mov ah, [bx - 1] ; Записываем значение текущего символа в ah (-1, потому что вначле сделали +1)

inc bx ; Увеличиваем di на 1, чтобы посмотреть следующий символ в исходной строке

dec di ; Уменьшаем di на 1 (идем по выходной строке назад)

mov [di], ah ; Записываем 'обратный' символ

jmp REVERSE

LOAD\_DI:

mov di, dx ; Востанавлием значение di

UPDATE:

cmp al, '$' ; Если конец строки, то записываем

je WRITE

mov IS\_SEP\_INVERSE, 0 ; Устанавлием флаг инверсивного сепаратора

mov LAST\_SEP\_INDEX, si ; Сохраняем адресс текущего сепаратора

cmp al, '/' ; Если это не инверсивный сепаратор, то идем дальше по циклу

je NEXT

mov IS\_SEP\_INVERSE, 1 ; Иначе уставливаем флаг в 1 и идем дальше по циклу

jmp NEXT

WRITE:

stosb ; Записываем символ из al в выходную строку и увеличиваем di

cmp al, '$' ; Проверка на конец строки

je OUTPUT ; Завершение, если достигнут конец строки

NEXT:

jmp ITERATION ; Переход к следующей итерации

OUTPUT:

mov dx, OFFSET STR2 ; Загружаем начало выходной строки

call WriteMsg ; Печатаем

mov COMPLETE\_FLAG, 1 ; Устанавлием флаг, что прерывание уже было выполненено

END\_PROCESS:

mov al, 20H

out 20H, al

; Восстановление содержимого регистров

pop di

pop si

pop es

pop ds

pop dx

pop bx

pop ax

iret

ProcessString ENDP

; Головная процедура

Main PROC FAR

push ds ;\ Сохранение адреса начала PSP в стеке

sub ax,ax ; > для последующего восстановления по

push ax ;/ команде ret, завершающей процедуру.

mov ax,DATA ; Загрузка сегментного

mov ds,ax ; регистра данных.

; Сохранение

mov ah, 35h ; функция получения вектора

mov al, 4bh ; номер вектора

INT 21H

mov KEEP\_IP, bx ; запоминание смещения

mov KEEP\_CS, es ; и сегмента

; Замена 4bh на ProcessString

push ds

mov dx, OFFSET ProcessString ; смещение для процедуры в dx

mov ax, SEG ProcessString ; сегмент процедуры

mov ds, ax ; помещаем в ds

mov ah, 25h ; функция установки вектора

mov al, 4bh ; номер вектора

int 21h ; меняем прерывание

pop ds

mov dx, OFFSET START\_MESSAGE ; Печать начального сообщения

call WriteMsg

mov dx, OFFSET STR1HEAD ; вызов функции ввода строки

call StringRead

mov dx, OFFSET NEWLINE ; Печать новой строки

call WriteMsg

int 4bh ; Вызов 4bh 2 раза

int 4bh

push ds

mov dx, KEEP\_IP ; смещение для оригинального обработчика 4ир

mov ax, KEEP\_CS ; сегмент оригинального обработчика

mov ds, ax ; помещаем в DS

mov ah, 25h ; функция установки вектора прерывания

mov al, 4bh ; номер вектора

int 21h ; восстанавливаем вектор

pop ds

ret

Main ENDP

CODE ENds

END Main