**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №5**

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

**Тема: Написание собственного прерывания.**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1304 |  | Чернякова В.А. |
| Преподаватель |  | Кирьянчиков В.А. |

Санкт-Петербург

2022

## Цель работы.

Изучить, как работают прерывания. Написать собственное прерывание.

## Задание.

В соответствии с 8 вариантом шифр задания – 2a, где

2 – 60h – прерывание пользователя – должно генерироваться в программе;

a – выполнить вывод сообщения на экран заданное число раз, после чего вставить фиксированную задержку и вывести сообщение о завершении обработчика.

## Ход работы.

В сегменте данных *DATA* содержится две переменных для хранения старого прерывания, содержавшегося по смещению *60h*, – *old\_seg, old\_ip.* Также в этом сегменте содержится *out\_msg* – сообщение, которое будет выводиться во время работы прерывания, *end\_msg* – сообщение, которое будет выведено после завершения работы прерывания.

*DATA SEGMENT*

*old\_seg dw 0;*

*old\_ip dw 0;*

*out\_msg DB 'Hello!$';*

*end\_msg DB 'End!$';*

*DATA ENDS*

В сегменте стека *AStack,* как и требуется по заданию, выделяется 1Кбайт памяти, то есть *DW 512.*

*AStack SEGMENT STACK*

*DW 512 DUP(?)*

*AStack ENDS*

В сегменте кода сначала определяется процедура пользовательского прерывания *CUSTOM\_INT.*

Сначала на стеке сохраняются значения регистров до входа в прерывание.

*push ax*

*push bx*

*push cx*

*push dx*

Для вывода сообщений используем функцию 9h, которая выводит на стандартное устройство вывода (например, на монитор) строку символов. Адрес строки предварительно записан в ds:dx. Строка должна ОБЯЗАТЕЛЬНО заканчиваться символом $ (24h), который не будет выведен в STDOUT.

*mov ah, 9h*

С помощью метки *print\_loop* строка из *ds:dx* выводится заданное в *cx* количество раз. С помощью прерывания *int 21h* происходит вызов функций *DOS*. Инструкция *loop* осуществляет работу цикла до тех пор, пока значение в *cx* не равно нулю. Параллельно при работе значение cx уменьшается на единицу.

*print\_loop:*

*int 21h;*

*loop print\_loop;*

Пауза после вывода строк реализуется при помощи прерывания *int 1Ah*, которое позволяет работать с таймером*.* С помощью mov ah, 0 получаем системное время. При вызове прерывания в регистре *bx* должна содержаться требуемая задержка (в тиках процессора). К требуемой задержке прибавляется текущее время в программе, которое прерыванием *1Ah* записывается в *cx, dx* (в *cx* – старшая часть значения).

*mov ah, 0;*

*int 1Ah;*

*add bx, dx;*

Далее в цикле происходит сравнение значения *bx* с текущим временем программы, если оно больше времени в *bx*, то производится выход из цикла. Инструкция *jg* позволяет выйти из цикла, переход на соответствующую метку происходит, если первый операнд больше второго. И при помощи прерывания *21h* происходит вывод строки, сообщающей о завершении работы прерывания. Оно хранится по адресу *ds:offset end\_msg.*

*cmp bx, dx;*

*jg pause;*

*mov dx, offset end\_msg;*

*mov ah, 9h*

*int 21h*

Далее перед выходом из прерывания восстанавливаются регистры из стека.

*pop dx*

*pop cx*

*pop bx*

*pop ax*

Также осуществляется выход из прерывания с помощью *20h* и *iret* – возврат из прерывания*.*

*mov al, 20h;*

*out 20h, al;*

*iret;*

Вызов прерывания происходит в процедуре Main.

Вначале объявляется значение регистров.

*push ds*

*sub ax, ax*

*push ax*

*mov ax, DATA;*

*mov ds, ax;*

С помощью прерывания *21h* получается прерывание, хранящееся по смещению 60h. Именно благодаря *35* мы получаем вектор прерывания по указанному после значения смещению. Старое прерывание сохраняется в переменных *old\_seg, old\_ip*.

*mov ax, 3560h;*

*int 21h*

*mov old\_seg, es;*

*mov old\_ip, bx;*

Далее также при помощи прерывания *21h* происходит запись по смещению *60h* нового прерывания *CUSTOM\_INT.*

*push ds*

*mov dx, offset CUSTOM\_INT*

*mov ax, seg CUSTOM\_INT*

*mov ds, ax*

*mov ax, 2560h*

*int 21h*

*pop ds*

Далее задаются значения регистров: в *ds:dx* должна лежать выводящаяся несколько раз строка, в *cx* – количество раз сколько нужно вывести строку, в *bx* – время задержки(в тиках процессора), в  *ds:offset* – сообщение о завершении.

*mov dx, offset out\_msg*

*mov cx, 10h*

*mov bx, 36h*

*int 60h*

После вызова нового прерывания происходит восстановление старого прерывания и выход из программы. CLI – cброс флага прерывания. STI – установка флага прерывания.

*CLI*

*push ds*

*mov dx, old\_ip*

*mov ax, old\_seg*

*mov ds, ax*

*mov ax, 2560h*

*int 21h*

*pop ds*

*STI*

Исходный код программы см. в приложении.

## Тестирование.

1. При вызове прерывания заданы следующие регистры:

ds:dx out\_msg (где out\_msg – это «Hello!»)

cx = 05h (количество повторов выводимых сообщений – 5, переведённое в 16-ричную систему счисления)

bx = 72h (время задержки в тактах процессора)

ds:offset end\_msg (где end\_msg – это «End!»)



Рисунок 1 – работы программы согласно пункту 1.

1. При вызове прерывания заданы следующие регистры:

ds:dx out\_msg (где out\_msg – это «Hello!»)

cx = 09h (количество повторов выводимых сообщений – 9, переведённое в 16-ричную систему счисления)

bx = 36h (время задержки в тактах процессора)

ds:offset end\_msg (где end\_msg – это «End!»)



Рисунок 2 – работа программы согласно пункту 2.

## Вывод.

В ходе выполнения лабораторной работы изучены виды прерываний и работа с ними. В соответствие с заданием создано собственное прерывание. Написана программа, выводящая строку заданное количество раз, после выставляющая задержку на заданное время и выводящая завершающее сообщение.

# ПРИЛОЖЕНИЕ.

Исходный код программы.

DATA SEGMENT

old\_seg dw 0;old interruption 60h

old\_ip dw 0;old interruption 60h

out\_msg DB 'Hello!$';message all interruption

end\_msg DB 'End!$';message end interruption

DATA ENDS

AStack SEGMENT STACK

DW 512 DUP(?)

AStack ENDS

CODE SEGMENT

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

CUSTOM\_INT PROC FAR;user interrupt procedure

;storing registers

push ax

push bx

push cx

push dx

mov ah, 9h;print string cx times

print\_loop:

int 21h;calling the DOS function manager

loop print\_loop;loop while cx != 0 cx = cx - 1

;pause

mov ah, 0;buffer adress

int 1Ah;work with timer

add bx, dx;add to the required delay current time in program

pause:

mov ah, 0 ; 1)заменить на mov ah,1

int 1Ah

cmp bx, dx;compare bx and current programm time

jg pause; bx > dx

;printing end message

mov dx, offset end\_msg;end\_msg output

mov ah, 9h

int 21h ; 2)удалить команду

;restoring registers

pop dx

pop cx

pop bx

pop ax

;return

mov al, 20h;end programm

out 20h, al;output data

iret;return from interruption

CUSTOM\_INT ENDP

Main PROC FAR

push ds

sub ax, ax

push ax

mov ax, DATA; ax = DATA

mov ds, ax; ds = ax

;storing old int

mov ax, 3560h;give interrupt 60h vector

int 21h

mov old\_seg, es;old\_seg = es ; 3)заменить на mov old\_seg, ds

mov old\_ip, bx;old\_ip = bx

;setting custom int

push ds

mov dx, offset CUSTOM\_INT

mov ax, seg CUSTOM\_INT

mov ds, ax ;4)заменить на mov es, ax

mov ax, 2560h

int 21h

pop ds

;setting registers according to custom int manual

mov dx, offset out\_msg

mov cx, 09h;number of messages

mov bx, 36h;delay in ticks of process /seconds/

int 60h

;restoring old int

CLI;resetting the interrupt flag

push ds

mov dx, old\_ip

mov ax, old\_seg

mov ds, ax

mov ax, 2560h

int 21h

pop ds ;5)удалить команду

STI;set interrupt flag

ret

Main ENDP

CODE ENDS

END Main

**Вопросы для защиты.**

*Внести указанные изменения (ошибки) 1-5 в код, рассматривая их по отдельности. Указать, почему и как они проявятся. Если можно, показать тесты, которые эти ошибки обнаружат и их результаты.*

# Защита

1)Заменить mov ah, 0 на mov ah,1

Команда MOV копирует данные из операнда-источника в операнд-получатель.

АХ делится на AH и AL и т.д. Буква "Н" означает верхний регистр.

АН в DOS программах используется как определитель, какой сервис будет использоваться при вызове INT.

Для работы с таймером (точнее говоря, для работы с каналом 0 таймера) BIOS содержит две функции прерывания INT 1Ah. Они позволяют прочитать текущее содержимое счетчика и изменить его.

Функция 00h предназначена для чтения содержимого счетчика таймера.

Мы заменяем данную функцию на 01h, которая изменяет содержимое счетчика таймера.

ТЕСТРИРОВАНИЕ.

**Ожидаемый результат:**



**Полученный результат:**



Программа выводит необходимое количество сообщений, за исключением финального сообщения обработчика и не заканчивает работу. Для перехода по метке jg pause сравнивалось количество тиков процессора, получаемых с помощью mov AH,0, значение сохранялось в cx:dx, с заданной задержкой, которая находилась в bx. Функция 01H же устанавливает часы – имеет только вход, без выходных данных. Из-за этого сравнение всегда будет срабатывать переход по метке jg pause, значит, программа не завершает обработку и финальное сообщение не выводится.

2)Удалить команду Int 21h

Все функции DOS вызываются с помощью прерывания 21h

ТЕСТИРОВАНИЕ.

**Ожидаемый результат:**



**Полученный результат:**



Программа завершает работу, выводит нужное количество сообщений, но не выводит финальное, об окончании работы обработчика. Объясняется это тем, что из-за отсутствия объявления Int 21h, отвечающего за вызов функций DOS, функция 09h, которая выводит строку символов на экран, не срабатывает и финальное сообщение на экране не появляется.

3)Заменить mov old\_seg,es на mov old\_seg, ds

Регистр сегмента данных (DS) содержит начальный адрес сегмента данных.

Регистр ES. Некоторые операции над строками используют дополнительный сегментный регистр для управления адресацией памяти.

ТЕСТИРОВАНИЕ.

**Ожидаемый результат:**



**Полученный результат:**



Программа завершает работу, ожидаемый и полученный результат совпадают. Происходит некорректное запоминание старого прерывания. Так как мы изменяли ds(mov ds, ax), соответственно в старый сегмент нашего перрывания будет неверный, правильнее будет написать mov old\_seg,es.

4)Заменить mov ds,ax на move es,ax

ТЕСТИРОВАНИЕ.

**Ожидаемый результат:**



**Полученный результат:**



Программа выводит правильные данные, но не заканчивает работу. В ax находится адрес сегмента с указанной переменной (mov ax, seg CUSTOM\_INT), соответственно передавать это адрес нужно в регистр, отвечающий за работу с сегментом данных – ds. Из-за передачи в регистр es программа не завершает корректно работу.

5)Удалить команду pop ds

ТЕСТИРОВАНИЕ.

Ожидаемый результат:



Полученный результат:



Программа выводит правильно данные, но не завершает работу. Это связано с тем, что команда pop осуществляет извлечение операнда из стека. Благодаря pop ds осуществляется загрузка слова из вершины стека в регистре DS. Перед этой командой прописан push ds – помещение сегмента данных в вершину стека. То есть мы заносим данные, но не забираем их. Из-за этого происходит некорректное использование памяти – программа не завершает работу, стек не пуст.