МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра вычислительные системы и технологии

Лабораторная работа № 1

Разработка многомодульных программ с использованием ассемблера и языков высокого уровня, сложные приемы программирования на ассемблере

ОТЧЕТ

по лабораторной работе

по дисциплине

Принципы и методы  
организации системных программных средств

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Викулова Е.Н.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сапожников В.О.

19-ИВТ-3

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2021

**Цель работы**

* Приобретение навыков разработки программ на ассемблере, перехватывающих аппаратные и программные прерывания в реальном режиме процессора;
* Получение навыков разработки многомодульных приложений (с использованием ассемблера и языка высокого уровня);
* Изучение принципов работы ОС Windows*,* разработка консольных и графических приложений Windows.

**Задание 1**

Разобрать тестовый пример или разработать на ассемблере программу, реализующую рекурсивный алгоритм, изучить работу стека.

В отчет поместить текст программы и скриншоты отладчика с пояснениями о работе стека при рекурсивном вызове

**Теория**

Регистр bp – ссылается на память относительно регистра стека ss

Регистр sp – ссылается на “вершину” стека – место в памяти куда будет положено следующее значение, загруженное в стек.

Для обращения к параметрам внутри процедуры обычно используют регистр BP. В самом начале процедуры содержимое регистра BP сохраняется в стеке и в него копируется значение регистра SP. Это позволяет «запомнить» положение вершины стека и адресовать параметры относительно регистра BP.

push bp

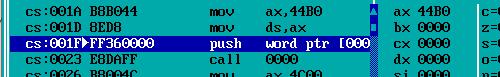
mov bp,sp

Команда mul <число> выполняет умножение al, если число – байт (ax если число - слово) на <число>

Команда jcxz передает управление по адресу, если значение в регистре cx равно нулю.

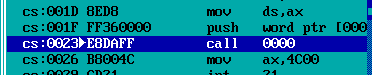
Команда ret выполняет выход из процедуры – передает управление по адресу на вершине стека.

**Разбор программы**

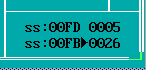


Заносим в стек число из памяти *ff = 5* – передача параметра в процедуру. ***Произошел первый занос значения в стек***





Вызываем процедуру факториала при это в стек попало значение адреса памяти, куда программа выйдет после завершения процедуры (команда ret без параметра возьмёт из стека значение, на которое указывает ss:sp и передаст его в cs:ip). ***Произошел второй занос значения в стек***





В самом начале процедуры содержимое регистра BP сохраняется в стеке и в него копируется значение регистра SP. Это позволяет «запомнить» положение вершины стека и адресовать параметры относительно регистра BP (mov cx, [bp + 04]). ***Произошел третий занос значения в стек***

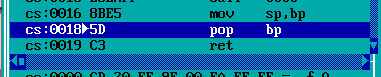


Это было последняя операция заноса значений в стек при вызове и работе процедуры. Таким образом за “одну итерацию” рекурсивной функции подсчета факториала в стек заносятся 3 значения: аргумент, адрес выхода, значение bp

Рассмотрим **ВЫХОД ИЗ РЕКУРСИИ** и что в этот момент происходит со стеком.

Значение стека на момент начала выхода из рекурсии:





Сначала в bp передается значение bp с прошлой итерации, т.е. теперь мы будет адресоваться от bp как на прошлой итерации, что позволит нам взаимодействовать со значениями, занесенными в стек на прошлой итерации.

***Произошло первое взятие значения из стека***





Затем команда ret передает управление (помещаем в cs:ip) значение из стека. ***Произошло второе взятие значения из стека***



На этом все операции взятия значений из стека на итерации закончились, т.е. за итерацию мы заносим в стек 3 значения, а забираем лишь 2. Таким образом в стеке остается значение переданного параметра, через который мы благополучно “перешагиваем” при помощи следующей команды:



И того за время работы процедуры подсчитывания факториала числа 5 в стеке **останется 5 значений**: 5, 4, 3, 2, 1

*Но т.к. sp сместился, то данные значения будут затерты при дальнейшей работе со стеком.*

**Листинг программы**

.MODEL SMALL

.STACK 0FFH

.DATA

ff dw 5

nn dw 1

.CODE

fact proc

push bp

mov bp,sp

mov cx,[bp+4]

mov ax,cx

mul nn

mov nn,ax

dec cx

jcxz end\_p

push cx

call fact

end\_p: mov sp,bp

pop bp

ret

fact endp

main: mov ax,@data

mov ds,ax

push ff

call fact

mov ax,4c00h

int 21h

end main

**Задание 2**

*Программные прерывания* вызывает непосредственно программа при помощи команды int (отсюда и название — программные).

*Аппаратные прерывания* вызываются самостоятельно процессором (аппарату-рой компьютера) при возникновении каких-либо событий. При этом процессор прекращает выполнение текущей программы, сохраняет в стеке регистры ss, sp и флаги, вызывает соответствующее прерывание, а затем восстанавливает сохраненные регистры и продолжает выполнение текущей программы.

Выполним перехват программного прерывания int 16. При “стандартном режиме” работы прерывание int 16h выполняет вызов клавиатурных функций BIOS:

00h – прочитать символ с клавиатуры

01h – получить состояние клавиатуры

02h – получить состояние флагов клавиатуры

03h – управление режимом автоповтора

04h – вкл/выкл звуковой сигнал клавиш

05h – поместить символ в буфер клавиатуры

10h – прочитать символ с расширенной клавиатуры

11h – получить состояние расширенной клавиатуры

12h – получить состояние флагов расширенной клавиатуры

Перехватив прерывание, мы можем заменить выполнение данных функций на выполнение нужного нам кода, например вызвать процедуру, которая внутри себя выполнит какую-то логику.

Для перехвата прерывания необходимо поместить адрес процедуры векторов по смещению n\*4.

В нашей программе мы реализуем перехват при помощи функции *DOS* 25*h*, 35*h.*

Функция **35***h* прерывание **21***h*  - считать адрес обработчика.

Вход:  ***ah***=35*h* - № функции;

***al***– № обработчика.

Возвращаемое значение: дальний адрес обработчика в ***es:bx.***

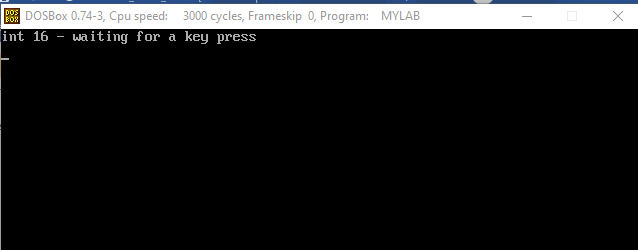
Функция **25***h* прерывание **21***h*  - установить обработчик.

Вход:  ***ah***=25*h* - № функции;

***al***– № обработчика;

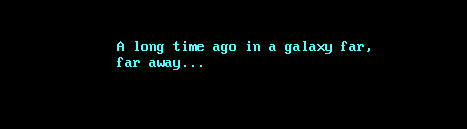
***ds:dx*** -дальний адрес обработчика.

Наше перехваченное прерывание будет выводить “вступление” лабораторной работы №4 прошлого семестра. В конце программы мы вернем старый обработчик прерываний на место.

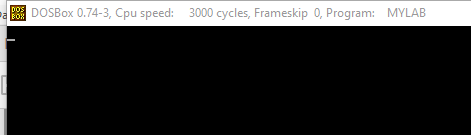


Стандартная работа прерывания – ожидание нажатия клавиши

Затем мы перехватываем прерывание, и оно выполняет код процедуры, которую мы записали в вектор







Ну и в конце вернем прерывание в норму. Теперь оно снова ожидает нажатие клавиши (Побаловались и хватит).

**Листинг программы**

;перехватим int 16h

;программа типа .com

.model tiny

.code

org 100h

;Макрос рисования окна

; xStart - левый верхний угол - столбец

; yStart - левый верхний угол - строка

; xEnd - правый нижний угол - столбец

; yEnd - правый нижний угол - строка

drawWindow macro xStart, yStart, xEnd, yEnd, color

mov ah, 06

mov al, 00

mov ch, yStart ;левый верхний угол - строка

mov cl, xStart ;левый верхний угол - столбец

mov dh, yEnd ;правый нижний угол - строка

mov dl, xEnd ;правый нижний угол - столбец

mov bh, color ;установка цвета фона и цвета букв

int 10h ;прерывание отрисовки

endm

;Макрос вывода в окне

; string - текст для вывода

; row - строка вывода

; column - колонка вывода

printInWindow macro string, row, column

push ax

push dx

mov ah,2

mov dh,row

mov dl,column

mov bh,0

int 10h

mov ah, 09h

mov dx, offset string

int 21h

pop dx

pop ax

endm

;Макрос ожидания при помощи функции 86h прерывания Int 15h

; time - время в миллисекундах

sleep macro time

mov al, 0

mov ah, 86h

mov cx, time

int 15h

endm

start:

mov ax,0003h

int 10h

mov ah,9

mov dx,offset mess1

int 21h

; вызовем стандартный обработчик int 16h

mov ah,0

int 16h

; функция 35h - считать вектор, № вектора в al

mov ax,3516h

int 21h

;сохраним вектор

mov [old\_16],bx

mov [old\_16+2],es

;установим наш обработчик: функция 25h, адрес обработчика в ds:dx

mov ax,2516h

mov dx,offset new\_16

int 21h

; обработчик установлен, вызовем его

mov ah,0

int 16h

;наш обработчик выполнился

; вернем старый на место

mov ax,2516h

mov dx,old\_16

mov bx,[old\_16+2]

mov ds,bx

int 21h

mov ah,0

int 16h

mov ax,4c00h

int 21h

;наш обработчик

new\_16 proc

push ds

push es

push ax

mov ax, videoSeg

mov es, ax

;\*\* Вывод фразы: a long time ago in a galaxy far far away \*\*\*\*\*

drawWindow 0, 0, 80, 25, startBGColor ;на всю консоль черное окно, голубые буквы

printInWindow LongTimeAgo, 11, 24 ;вывод фразы в данном окне

call hideCursor ;прячем курсор

sleep 85 ;ождиание

mov ax, 03h ;очистка экрана

int 10h

drawWindow 0, 0, 80, 25, titleBGColor ;на всю консоль черное

;окно, желтые буквы

printInWindow Separator, 11, 34 ;вывод

printInWindow Assembler, 12, 35

printInWindow Separator, 13, 34

call hideCursor

sleep 75 ;ожидание

mov ax, 03h ;очистка экрана

int 10h

pop ax

pop es

pop ds

iret

new\_16 endp

;Процедура прятания курсора

;устанавливает курсор за пределами окна

hideCursor PROC

mov ah,2 ;прячем курсор

mov dh,26 ;устанавливаем его за пределы экрана

mov dl,81

mov bh,0

int 10h

ret

ENDP

; данные

old\_16 dw ?,?

mess db 'old int 16 ne rabotaet!!!$'

n=$-mess

mess1 db 'int 16 - waiting for a key press',10,13,'$'

startBGColor equ 00001011b ;черный фон - голубые буквы

videoSeg equ 0b800h

;Давно в далекой далекой галактике....

LongTimeAgo db 'A long time ago in a galaxy far,',10, 24 dup (' '), 'far away...$'

;Ассемблерные войны

Assembler db 'ASSEMBLER$',10,13

Separator db '===========$', 10, 13

titleBGColor equ 00001110b ;черный фон - желтые буквы

end start

**Задание 3**

Резидентная программа – программа, возвращающая управление ОС, но остающаяся в ОП.

Резидентная программа состоит из 2-х частей:

* инициализирующей – с нее начинается выполнение; перехватывает прерывания и завершается; выгружается, оставляя резидентную часть;
* резидентной – та, которая осталась в памяти; ее адрес записан в таблицу векторов обработчиков прерывания; активизируется всякий раз, когда возникает прерывание, которое было перехвачено.

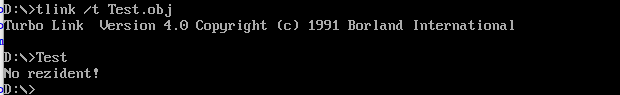
**Защита**

Поскольку резидентная программа оставляет после себя след в памяти, то с каждым ее запуском в памяти может скапливаться все больше и больше мусора. Есть несколько способов проверить наличие резидентной программы.

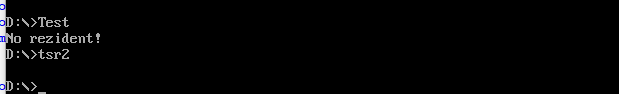
Воспользуемся самым простым из них: в резидентной части по определённому адресу запишем константану. В качестве адреса выберем адрес резидента – 2. Теперь при запуске программы будем обращаться к тому адресу и проверять данные с заданным значением. Если они совпадают – резидент уже загружен.

**Цель**

В данном программе мы будем перехватывать прерывание 21h и влиять на функцию 9h. Когда будет вызваться функция 9h 21ого прерывания для вывода строки, мы будем подменивать строку, которую необходимо вывести на свою – “Boo, it is Resident Evil!!!”



Вызов тестовой программы до загрузки резидента



Загрузка резидента



Запуск тестовой программы после загрузки резидента

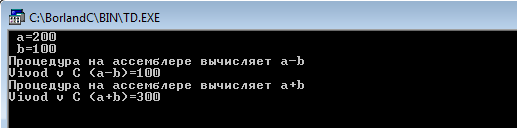


Защита от повторной загрузки резиденте

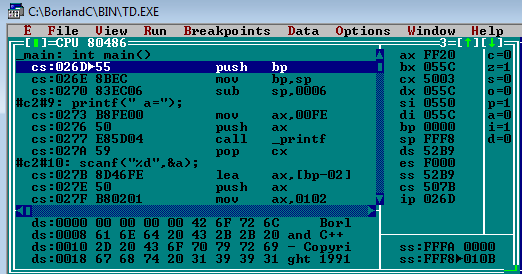
**Задание 4**

***Конвенции*** о передаче параметров определяют:

* + порядок занесения параметров в стек (прямой, обратный);
  + очистку стека (стек чистит вызывающая/вызываемая процедура);
  + искажение имен внешних процедур.
* Конвенция ***Pascal* –** параметры помещаются в стек в прямом порядке (т.е. в том порядке, в котором они следуют в объявлении функции, начиная с первого аргумента), очистку стека выполняет вызываемая процедура.
* Конвенция ***С*** – параметры помещаются в стек в обратном порядке (начиная с последнего аргумента в объявлении функции), очистку стека выполняет вызывающая процедура.

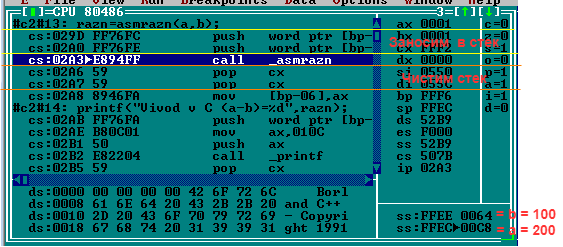


Результат работы программы



Запуск c\_asm через отладчик BorlandC.

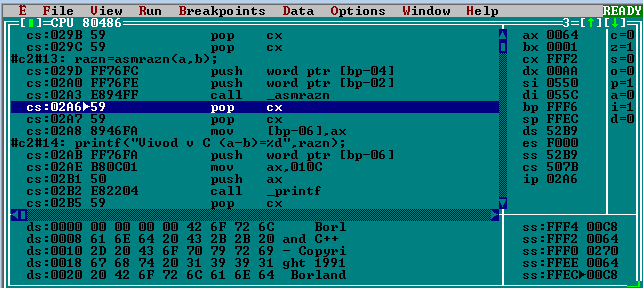
Можно заметить, что функции printf и scanf работают по С конвенции т.к. очистка стека происходит вызываемое процедурой, а не внутри самой функции.



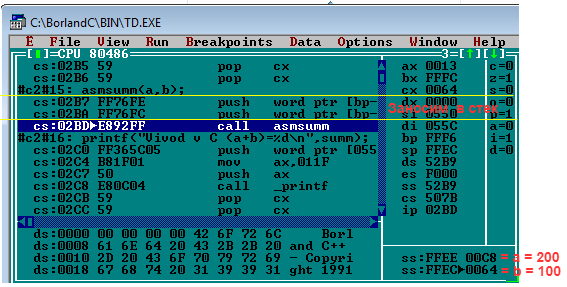
extern "C" cdecl int asmrazn(int,int);

Т.к. мы явно указали что функция asmarzn имеет конвенцию С, то параметры в нее заносятся в обратном порядке: вводились сначала a, затем b; в функцию передается сначала b, затем а.





Так же поскольку данная функция имеет С конвенцию очистка стека происходит из вызывающей процедуры.



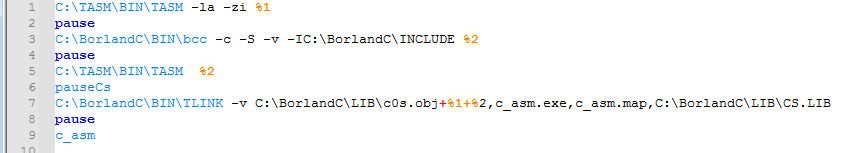
extern "C" pascal asmsumm(int,int);

Т.к. мы явно указали что функция asmarzn имеет конвенцию Pascal, то параметры в нее заносятся в прямом порядке: вводились сначала a, затем b; в функцию передается сначала a, затем b.



Так же поскольку данная функция имеет Pascal конвенцию очистка стека происходит из вызывающей функции.

Для сборки данной программы использовался файл link.bat со следующими параметрами



**Листинг программы**

**a1.asm**

.model small

extrn \_printf:near

extrn \_getch:near

extrn \_summ:word

PUBLIC \_asmrazn

PUBLIC asmsumm

.CODE

\_asmrazn PROC near ;x:word,y:word

push bp

mov bp,sp

;Вызов станд. функии С printf(mes)

mov dx,offset mes1

push dx

call \_printf

pop ax

mov ax,[bp+4] ;x

sub ax,[bp+6] ;y

;Функция возвращает значение через AX

push ax

call \_getch

pop ax

pop bp

ret

\_asmrazn endp

asmsumm PROC near x:word,y:word

push bp

mov bp,sp

;Вызов станд. функии С printf(mes)

mov dx,offset mes2

push dx

call \_printf

pop ax

mov ax,y ;[bp+4]

add ax,x ;[bp+6]

;Функция возвращает значение через \_summ

mov \_summ,ax

call \_getch

pop bp

ret 4

asmsumm endp

.data

mes1 db "Assembler: b-a ",10,13,0

mes2 db 10,13,"Assembler: a+b",10,13,0

end

**c2.cpp**

#include <stdio.h>

#include <conio.h>

extern "C" cdecl int asmrazn(int,int);

extern "C" pascal asmsumm(int,int);

int summ;

int main()

{

int a,b,razn;

clrscr();

printf("\nAt first C function\n");

printf("\nEnter two numbers:\n");

printf(" a=");

scanf("%d",&a);

printf(" b=");

scanf("%d",&b);

razn=asmrazn(a,b);

printf("\nFunction C: print the difference (a-b)=%d",razn);

asmsumm(a,b);

printf("\nFunction C: print the sum(a+b)=%d",summ);

printf("\nThe End of C function\n");

getch();

return(0);

}

**Задание 5**

**Первое консольное приложение** cons.asm

Структура программы представляет собой вызов функций - аргументы помещаются в стек, вызывается функция.

В программе осуществляется вызов трех системных функций:

запрос стандартных дескрипторов ввода-вывода - GetStdHandle

вывод строки в консоль - WriteConsoleA

завершение процесса - ExitProcess

Так же используется процедура lenstr, которая возвращает количество символов в строке, используемое в WriteConsoleA.

Трансляция и компоновка осуществляется следующим образом:

tasm32 /la /zi cons

ilink32 /Tpe /ap /v cons

/la - Показать в листинге код, вставляемый транслятором для организации интерфейса с языками высокого уровня.

/zi – Включить в объектный код информацию для отладки

/Tpe – создать exe файл

/ap – создать консольное приложение

/v - Поместить в исполняемый файл полную отладочную информацию.

**Листинг программы**

;консольное приложение: получить дескриптор вывода, вывести 2 строки

.586

;помещение параметров в стек справа налево, результат в eax, стек "чистит" вызывающая

.MODEL FLAT, STDCALL

STD\_OUTPUT\_HANDLE equ -11

EXTERN GetStdHandle:NEAR

EXTERN WriteConsoleA:NEAR

EXTERN ExitProcess:NEAR

includelib import32.lib

;---------------------------------------------------------------

\_DATA SEGMENT

str1 DB 10,13,"First Console Application",0

str2 DB 10,13,"Hello, world!",0

lens DD ? ;кол.введенных символов

res DD 0

HandleOut DD ?

\_DATA ENDS

\_TEXT SEGMENT

START:

;получить дескриптор стандартного потока вывода

push STD\_OUTPUT\_HANDLE

call GetStdHandle

mov HandleOut,eax

;длина строки

push offset str1

call LENSTR

;вывести строку

push offset res ; резерв

push offset lens ;выведено символов

push ebx ;длина строки

push offset str1 ;адрес строки

push eax ;HANDLE вывода

call WriteConsoleA

;\*\*\*\*\*\*\*\*\*\*\*\*

;длина строки

push offset str2

call LENSTR

;вывести строку

push offset res ; резерв

push offset lens ;выведено символов

push ebx ;длина строки

push offset str2 ;адрес строки

push HandleOut ;HANDLE вывода

call WriteConsoleA

push 0

call ExitProcess

;строка - [ebp+08h]

;длина в ebx

;

LENSTR PROC

push ebp

mov ebp,esp

push eax

push edi

;----------------------

cld

mov edi,dword ptr [ebp+08h]

mov ebx,edi

mov ecx,100

xor al,al

repne scasb ;найти символ 0

sub edi,ebx ;длина строки, включая 0

mov ebx,edi

dec ebx

;-----------------------

pop edi

pop eax

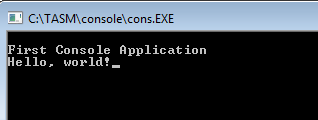
pop ebp

ret 4

LENSTR ENDP

\_TEXT ENDS

END START



**MessageBox** mb.asm

В программе осуществляется вызов двух системных функций:

отображение окна с сообщением - MessageBoxA

завершение процесса - ExitProcess

Трансляция и компоновка осуществляется следующим образом:

tasm32 mb

ilink32 /Tpe /aa mb

/Tpe – создать exe файл

/аа — создать обычное Windows-приложение

**Листинг программы**

includelib import32.lib

extrn MessageBoxA:near

extrn ExitProcess:near

.386

.model flat

.const

title1 db "First program",0

mess1 db "Clouse???",0

title2 db "Next",0

mess2 db "You must say Yes!!!",0

MB\_YESNO equ 4h

MB\_ICONINFORMATION equ 40h

IDYES EQU 6

IDNO EQU 7

.code

\_start:

push MB\_YESNO

push offset title1

push offset mess1

push 0

call MessageBoxA

cmp eax,IDYES

je m1

push MB\_ICONINFORMATION

push offset title2

push offset mess2

push 0

call MessageBoxA

m1: push 0

call ExitProcess

end \_start

