МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

ОТЧЕТ

по лабораторной работе №1

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

по дисциплине

«Принципы и методы организации системных

программных средств»

РУКОВОДИТЕЛЬ:

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

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сухоруков В.А.\_\_\_

(подпись) (фамилия, и.,о.)

\_\_\_\_\_\_19-В-2\_\_\_\_\_\_\_\_

(шифр группы)

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

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

Нижний Новгород 2021

Оглавление

[Цель работы 3](#_Toc97059716)

[Ход работы 3](#_Toc97059717)

[Задание 1 3](#_Toc97059718)

[Код программы с комментариями. 3](#_Toc97059719)

[Скриншоты отладчика 5](#_Toc97059720)

[Задание 2 8](#_Toc97059721)

[Код программы с комментариями. 8](#_Toc97059722)

[Скриншот работы программы 9](#_Toc97059723)

[Задание 3 10](#_Toc97059724)

[Код программы с комментариями. 10](#_Toc97059725)

[Скриншоты работы программ 14](#_Toc97059726)

[Задание 4 17](#_Toc97059727)

[Код программы с комментариями. 17](#_Toc97059728)

[Скриншот работы программы, скриншоты отладчика 19](#_Toc97059729)

[Задание 5 22](#_Toc97059730)

[Код программы с комментариями. 22](#_Toc97059731)

[Скриншоты работы программ 26](#_Toc97059732)

# 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.

# Ход работы

## Задание 1

Написать программу, которая с помощью рекурсии считает сумму цифр в числе.

### Код программы с комментариями.

;Программа вычисляет сумму цифр в числе с помощью рекурсии

;Начальное число записывается в переменную number

;Результат записывается в переменную summa

.MODEL SMALL

.STACK 0FFH

.DATA

number dw 4321

summa dw 0

.CODE

sum proc

**push** bp

**mov** bp,sp

;Получаем аргумент из стека

**mov** cx,[bp+4]

;Получаем последнюю цифру числа

**mov** ax,cx

**mov** cx,10

**div** cx

;Добавляем к сумме последнюю цифру

**mov** bx,summa

**add** bx,dx

**mov** summa,bx

;Проверяем число на наличие в нем цифр

**mov** cx,ax

**jcxz** end\_p

**mov** dx,0

;Заносим в стек оставшееся число

**push** ax

**call** sum

end\_p:

**mov** sp,bp

**pop** bp

**ret**

sum endp

main:

;Инициализация сегмента данных

**mov** ax,@data

**mov** ds,ax

**push** number

**call** sum

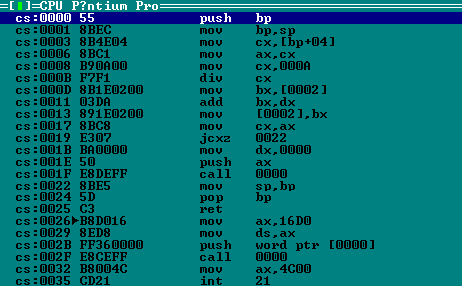
**mov** ax,4c00h

**int** 21h

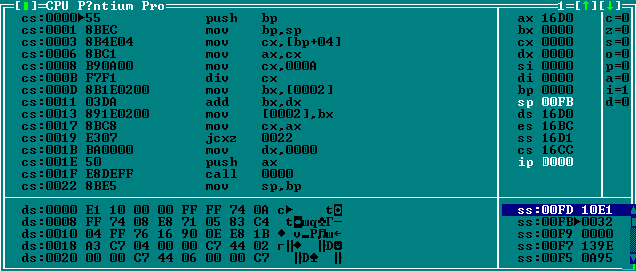
end main

### Скриншоты отладчика

Код программы, отображаемый отладчиком

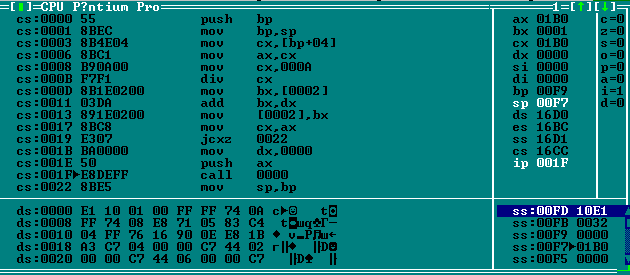


Помещаем в стек операнд number=4321d=10E1h. При вызове процедуры в стек помещается адрес возврата-0032h.



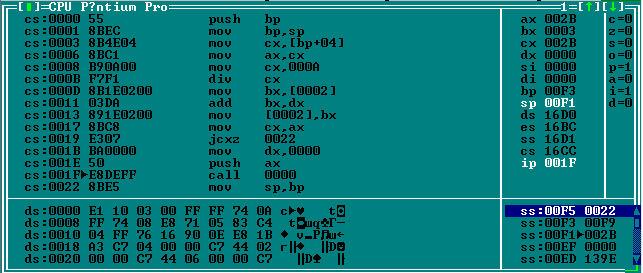
Помещаем операнд в регистр ax. В регистре dx будет вычисляться последняя цифра числа. Если цифры в числе не закончились процедура вызывается с новым операндом. В стек помещается значение регистра bp и новый операнд для процедуры

После первого выполнения процедуры в стек занесся новый операнд 01B0h=432d.

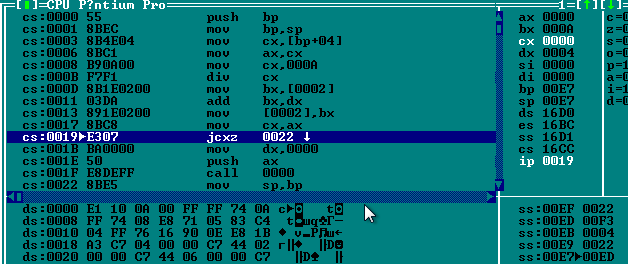


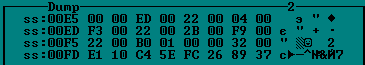
После второго выполнения процедуры в стек добавились:

1. Адрес возврата на первый вызов процедуры
2. Значение регистра bp
3. То, что осталось от числа на текущем этапе - 00B2h=43d.



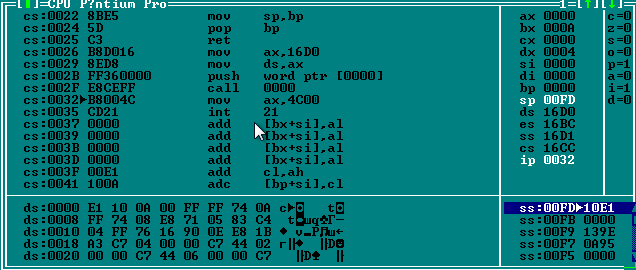
Когда от числа ничего не останется и значение суммы его цифр будет посчитано, начнется последовательное возвращение из процедур и удаление данных из стека.

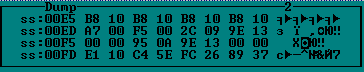




После возвращения из процедур к основной программе из него удалятся все занесённые в него данных.

Результат сложения цифр числа будет находится по адресу ds:02h, он будет равен 0Ah=10d=4+3+2+1.





## Задание 2

Изучить тестовый пример программы на ассемблере, которая перехватывает аппаратное прерывание.

В качестве тестового примера возьмём файл int8.asm. Написанная в нем программа предназначена для перехвата аппаратного прерывание- прерывание системного таймера.

### Код программы с комментариями.

Для перехвата прерывания необходимо в адресном пространстве стандартного обработчика прерывания поместить код того обработчика, который необходим.

Если известен «нужный» адрес, то достаточно разместить там свой код. Если адрес не известен, то его можно узнать.

; получим адрес стандартного обработчика

**mov** ax,3508h

**int** 21h ; vector v es: bx

**mov** **word** ptr old\_8, bx

**mov** **word** ptr old\_8+2, es

Для установки нового обработчика используется следующий код:

; установим новый обработчик, адрес в ds: dx

**mov** ax,2508h

**mov** dx, **offset** new\_8

**mov** cx, **seg** new\_8

**mov** ds, cx

**int** 21h

Чтобы убедиться, что обработчик работает вызовем функцию с ожиданием нажатия клавиши

**mov** ah,01

**int** 21h

После «баловства» необходимо восстановить стандартный обработчик.

**mov** ax,2508h

**lds** dx,old\_8

**int** 21h

Для нового обработчика напишем процедуру new\_8.

За обработку прерываний отвечает PIC (Programmable Interrupt Controller) – программируемый контроллер прерываний. PIC, передав в CPU сигнал int, блокирует внутри себя все IRQ (Interrupt Request) – каналы запросов прерываний, начиная с того, который вызвал прерывание и до последнего по возрастанию номера IRQ. Для снятия блокировки необходима команда EOI (End Of Interrupt) при завершении обработчика.

У данной команды нет обозначения, но есть код =20h. При завершении обработчика данный код пересылается в порт нужного PIC.

№ портов для EOI: порт ведущего - 20h, порт ведомого - A0h.

new\_8 **proc**

;вывод символа V на экран

**push** ax

**mov** ah,0eh

**mov** al,'V'

**int** 10h

;EOI

**mov** al,20h

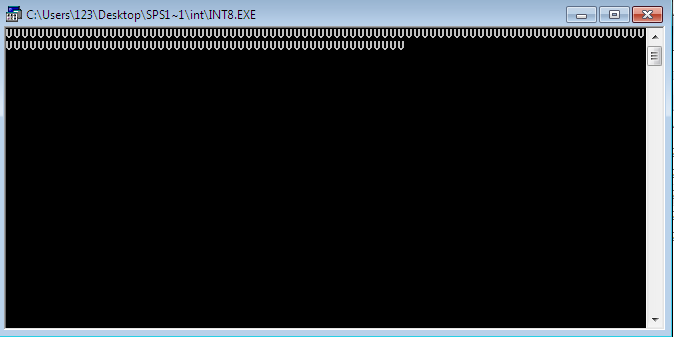
**out** 20h,al

**pop** ax

**iret**

new\_8 **endp**

### Скриншот работы программы



## Задание 3

Изучить тестовые примеры реализации резидентных программ.

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

Резидентная программа состоит из 2-х частей:

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

### Код программы с комментариями.

Рассмотрим первый пример – *tsr.asm*. Цель программы – заменить обработчик 21h прерывания для 9 функции.

Программа состоит из двух процедур – main **proc** и init **proc**. Работы программы начинается с процедуры main.

main **proc**

**jmp** init

; резидентные данные

; Адрес старого обработчика

old\_21h\_off **dw** ?

old\_21h\_seg **dw** ?

; Строка от резидента

string **db** 'Boo, it is Resident Evil!!!$'

; резидентный код

new\_21:

**cmp** ah,9

**je** ok\_09

; Переход на старый обработчик,

; все как обычно

**jmp** dword ptr cs:old\_21h\_off

ok\_09:

**push** cs

**pop** ds

; Замена при выводе любой строки на свою

**mov** dx,offset string **mov** ah,9

; старый обработчик выполнится,

; но строку мы подменили

**jmp** dword ptr cs: old\_21h\_off

main **endp**

Таким образом main proc – это новый обработчик 21h прерывания. После его установления в процедуре init все программы, который будут использовать 9 функцию вместе вывода требуемой строки будут выводить «пугающую» строчку.

Рассмотрим теперь процедуру init.

init **proc**

; получим вектор прерывания 21h es:[bx]

**mov** ax,3521h

**int** 21h

; но прежде чем изменить, сохраним его

; запишем старый адрес в old\_21h\_...

**mov** word ptr old\_21h\_off,bx

**mov** word ptr old\_21h\_seg,es

; Установим новый обработчик

**mov** ax,2521h

**mov** dx, offset new\_21 ; наш код

**int** 21h

; Завершить и оставить резидента

**mov** dx,(init-main+10FH)/16

**mov** ax,3100h

**int** 21h

;программа завершится, а резидент останется

;теперь при запуске любой программы, которая

;вызовет функцию 9 int21 будет выводиться "наша" строка

init **endp**

В процедуре стандартный обработчик заменяется новым обработчиком, как во второй задаче.

Для того, чтобы резидентная часть не пропала из оперативной памяти после завершения программы используется 31h функция 21h прерывания. Параметры для функции:

1. al – код возврата
2. dx – размер резидента в параграфах (параграф=16 байт)

Если запускать резидентную программу несколько раз, при каждом запуске в оперативной памяти будет резервироваться область для резидентной части.

Есть несколько способов решения этой проблемы. Рассмотрим файл *tsr2.asm*.

main **proc**

**jmp** init

; резидентные данные

; Адрес старого обработчика

old\_21h\_off **dw** ?

old\_21h\_seg **dw** ?

; Строка от резидента

string **db** 'Boo, it is Resident Evil!!!$'

;для проверки: установлен резидент или нет

sign **dw** 0AAAAh

; резидентный код

new\_21:

**...**

main **endp**

Процедура main отличается от предыдущей версии наличием новой переменной для проверки нахождения резидента в памяти - sign **dw** 0AAAAh. Рассмотрим теперь процедуру init.

init **proc**

; получим вектор прерывания 21h es:[bx]

**mov** ax,3521h

**int** 21h

; обратимся на 2 байта выше

**mov** ax, es:[bx-2]

; Проверяем находится ли резидент в памяти

**cmp** ax, 0AAAAh

; если да, повторно не загружаем

**je** no\_load

; запишем старый адрес в old\_21h\_...

**mov** word ptr old\_21h\_off,bx

**mov** word ptr old\_21h\_seg,es

; Установим новый обработчик

**mov** ax,2521h

**mov** dx, offset new\_21 ; наш код

**int** 21h

; Завершить и оставить резидента

**mov** dx,(init-main+10FH)/16

**mov** ax,3100h

**int** 21h

;мы уже в памяти, сообщим об этом

no\_load:

**mov** ah,40h

**mov** dx,offset test\_str

**mov** cx,nn

**mov** bx,1

**int** 21h

**mov** ax,4c00h

**int** 21h

init **endp**

test\_str db 'Your Resident already loaded!'

nn=$-test\_str

Для того чтобы повторно не загружать резидента в память проверяется есть ли на 2 байта выше обработчика кодовая фраза - 0AAAAh, которая должна быть у нашего обработчика.

Для проверки выполнения резидентной программы скомпилируем файл Test.asm:

;тестовая программа для экспериментов с резидентом

c1 segment

assume cs:c1, ds:c1, ss:c1

org 100h

start:

**mov** ah, 9

**mov** dx,offset mess\_in

**int** 21h

**mov** ax, 4c00h

**int** 21h

mess\_in db 'No rezident!$'

c1 ends

end start

### Скриншоты работы программ

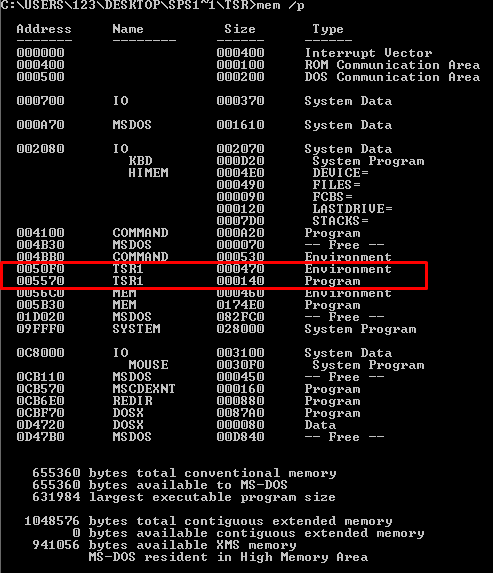
Примечание: программы tsr1, tsr2, tsr3, test написаны для com формата, поэтому при линковке нужно использовать ключ **/t**.



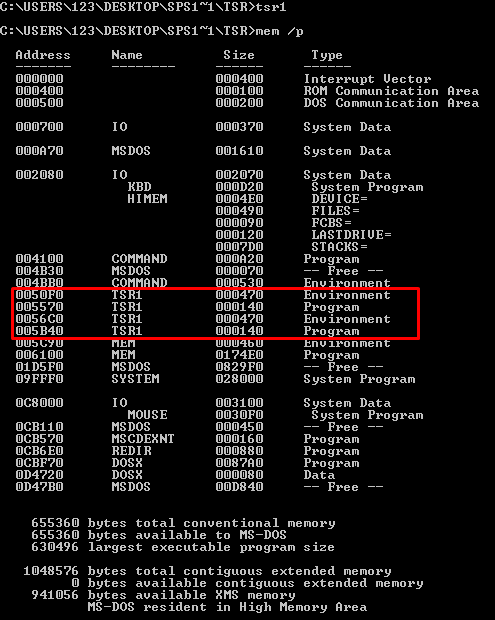
После запуска программы tsr1 любое использование во всех программах 9h функции 21h прерывания будет выводить строку «Boo, it is Resident Evil».



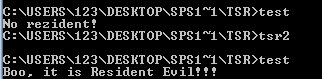
Использование команды **mem** с ключом **/p** позволяет убедиться, что резидентная часть осталась в памяти:

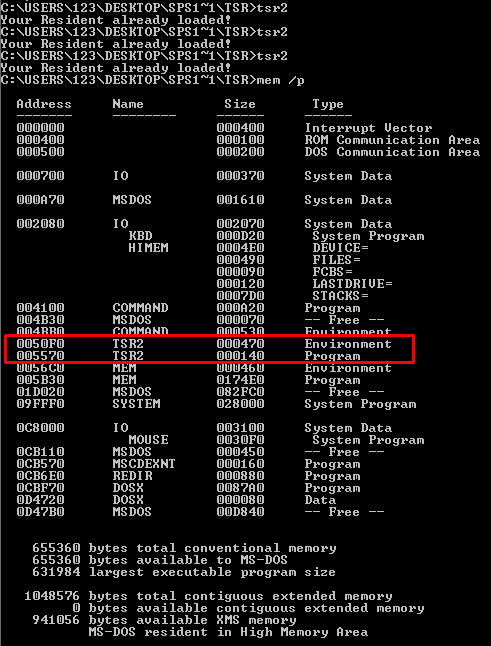


При повторном запуске tsr1 будет занимать дополнительные участки памяти.



При запуске tsr2 также, как и в случае с tsr1 строка будет заменяться, но повторное её использование не будет занимать новую память.





## Задание 4

Разобрать тестовый пример программы, состоящей из двух модулей: модуль на *C*, вызывающий несколько внешних функций и модуль на ассемблере с вызовами библиотечных функций.

### Код программы с комментариями.

Модуль на С представлен в файле c2.cpp.

/\*Подключение библиотек \*/

#include <stdio.h>

#include <conio.h>

/\*Подключение двух внешних функций \*/

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

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

/\*Глобальная переменная \*/

int summ;

int main()

{

/\*Локальные переменные \*/

int a,b,razn;

**printf**(" a=");

**scanf**("%d",&a);

**printf**(" b=");

**scanf**("%d",&b);

**razn**=asmrazn(a,b);

**printf**("\n‚Vivod v C (a-b)=%d",razn);

**asmsumm**(a,b);

**printf**("\n‚Vivod v C (a+b)=%d",summ);

**return**(0);

}

При подключении внешних функций следует указать способ передачи параметров: cdecl или pascal.

* 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.

Рассмотрим a1.asm.

.model small

extrn \_printf:near

extrn \_getch:near

extrn \_summ:word

PUBLIC \_asmrazn

PUBLIC asmsumm

.CODE

\_asmrazn **PROC** near

**push** bp

**mov** bp,sp

;Вызов функции С printf(mes1)

**mov** dx,offset mes1

**push** dx

**call** \_printf

**pop** ax

**mov** ax,[bp+4] ;1-ый аргумент a

**sub** ax,[bp+6] ;2-ой аргумент b, разность a-b

;Возвращаемое значение в AX

**push** ax

**call** \_getch

**pop** ax

**pop** bp

**ret**;вернуться, а стек очистит вызвавшая нас функция

\_asmrazn **endp**

;для разнообразия в следующей процедуре используем ;возможность определить аргументы в директиве PROC

;порядок – как в объявлении функции на ЯВУ:x-1-ый;y-2-ой

asmsumm **PROC** near x:word,y:word

**push** bp

**mov** bp,sp

;Вызов функции С printf(mes2)

**mov** dx,offset mes2

**push** dx

**call** \_printf

**pop** ax

**mov** ax,y ;2-ой аргумент b [bp+4]

**add** ax,x ;1-ый аргумент a [bp+6]и сумма

;Возвращаемое значение \_summ

**mov** \_summ,ax

**call** \_getch

**pop** bp

**ret** 4;вернуться и очистить стек (вытолкнуть 4 байта)

asmsumm **endp**

.data

mes1 db "Процедура на ассемблере вычисляет a-b ”,10,13,0

mes2 db 10,13,"Процедура на ассемблере вычисляет a+b”,10,13,0

end;

Функция asmrazn использует конвенцию С, asmsumm – конвенцию паскаль.

### Скриншот работы программы, скриншоты отладчика

Для трансляции и компановки используем файл link.bat.

TASM -la -zi %1

pause

bcc -c -S -v -IC:\BorlandC\INCLUDE %2

pause

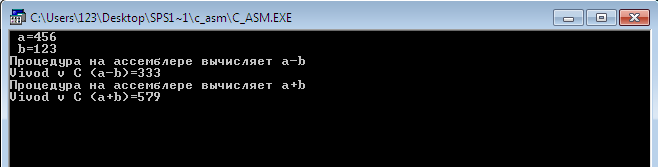
TASM %2

pause

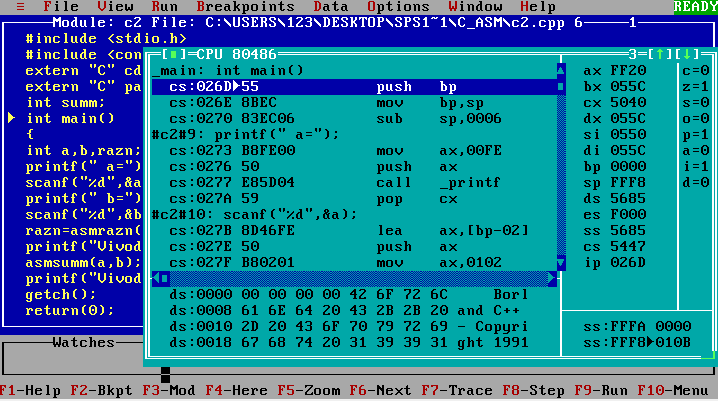
TLINK –v C:\BorlandC\LIB\c0s.obj+%1+%2,c\_asm.exe,c\_asm.map,C:\BorlandC\LIB\cs.lib

pause

c\_asm

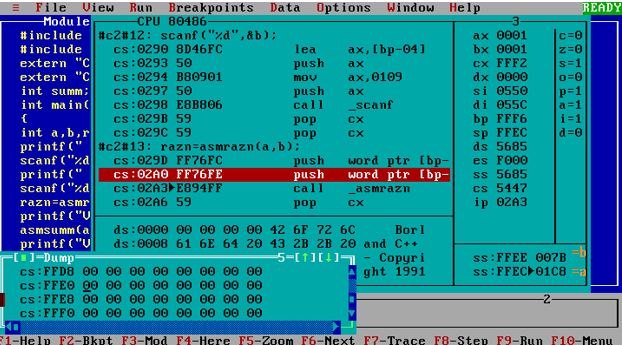


Запуск программы c\_asm.exe через сmd.exe.

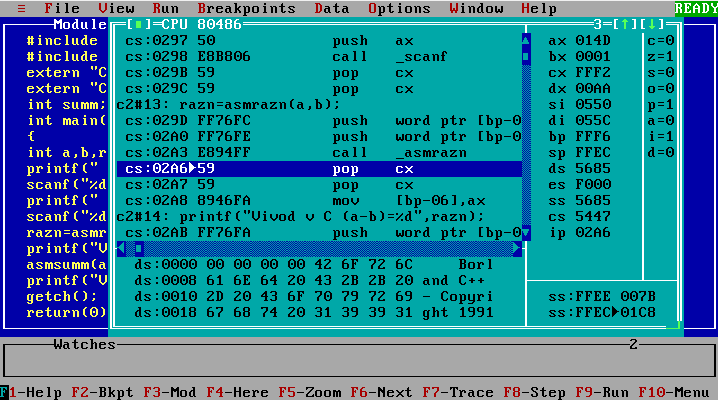


Запуск программы c\_asm.exe через td.exe.

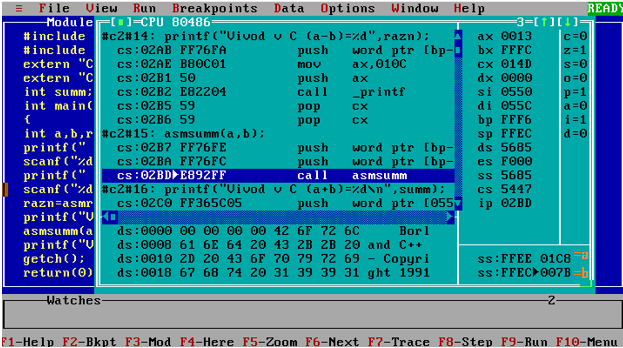




Процедура asmrazn передает параметры по конвенции С- в обратном порядке – сначала b=123d=7Bh, потом a=456d=1C8h.



После выполнения **call \_asmrazn** стек не отчистился, его отчистит вызывающая функция.



Процедура asmsumm передает параметры по конвенции pascal- в прямом порядке – сначала a=456d=1C8h, потом b=123d=7Bh. Очистка стека выполняется в вызываемой функции.



## Задание 5

Изучить тестовый пример консольного приложения Windows на ассемблере.

### Код программы с комментариями.

Рассмотрим простое консольное приложение, выводящее 2 строки cons.asm:

;консольное приложение: получить дескриптор вывода, вывести 2 строки

.586

.**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

**pop** 4

**LENSTR ENDP**

**\_TEXT ENDS**

**END START**

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.

Рассмотрим графическое приложение – файл mb.asm:

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**

В программе осуществляется вызов двух системных функций:

* отображение окна с сообщением - MessageBoxA
* завершение процесса - ExitProcess

Трансляция и компоновка осуществляется следующим образом:

tasm32 mb

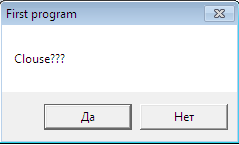
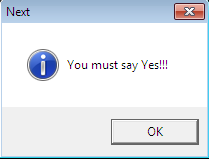
ilink32 /Tpe /aa mb

* /Tpe – создать exe файл
* /аа — создать обычное Windows-приложение

### Скриншоты работы программ



Cons.exe.

mb.exe