

Министерство науки высшего образования Российской Федерации  
федеральное государственное бюджетное образовательное

учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
им. Н. П. ОГАРЁВА»

(ФГБОУ ВО МГУ им. Н.П. Огарева)

Факультет довузовской подготовки и  
среднего профессионального образования

Выпускающая предметная цикловая комиссия общепрофессиональных и  
специальных (информационно-коммуникационных) дисциплин

## ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ 6

«Стек и простые процедуры»

по дисциплине: «Системное программирование»

Автор лабораторной работы \_\_\_\_\_ Н.С. Потапов  
(подпись) (дата)

Направление подготовки 09.02.03 Программирование в компьютерных  
системах

Преподаватель \_\_\_\_\_ В.В. Никулин  
(подпись) (дата)

Саранск

2023

## Выполнение задания 20:

Перед выполнением задания, изучил теоретический материал в тексте задания (Рисунки 1-2).

### Учебный курс. Часть 20. Стек

Стеком называется структура данных, организованная по принципу LIFO ("Last In – First Out" или "последним пришёл – первым ушёл"). Стек является неотъемлемой частью архитектуры процессора и поддерживается на аппаратном уровне: в процессоре есть специальные регистры (SS, BP, SP) и команды для работы со стеком. Обычно стек используется для сохранения адресов возврата и передачи аргументов при вызове процедур (о процедурах в следующей части), также в нём выделяется память для локальных переменных. Кроме того, в стеке можно временно сохранять значения регистров.

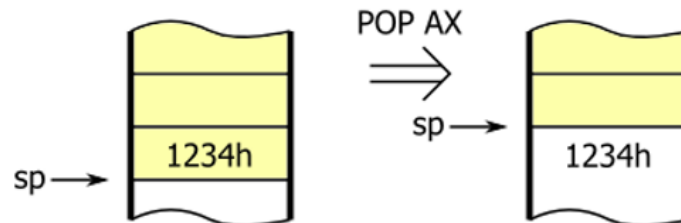
Рисунок 1 – Изучение материала

#### Извлечение элемента из стека

Выполняется командой **POP**. У этой команды также один операнд, который может быть 16-битным регистром (в том числе сегментом, но кроме CS) или 16-битной переменной в памяти. Команда работает следующим образом:

1. операнд читается из памяти по адресу в SP;
2. значение в регистре SP увеличивается на 2.

Обратите внимание, что извлеченный из стека элемент не обнуляется и не затирается в памяти, а просто остаётся как мусор. Он будет перезаписан при помещении нового значения в стек.



Примеры:

```
pop cx      ;Поместить значение из стека в CX
pop es     ;Поместить значение из стека в ES
pop [x]    ;Поместить значение из стека в переменную x
pop word [di] ;Поместить значение из стека в слово по адресу в DI
```

Соответственно, есть ещё 2 команды. **POPF** помещает значение с вершины стека в регистр флагов. **POPA** восстанавливает из стека все регистры общего назначения (но при этом значение для SP игнорируется).

Рисунок 2 – Изучение материала

После изучения материала приступил к рассматриванию примеров программ, запустил программу с помощью Turbo Debugger и посмотрел её исполнение (Рисунок 3-4).

```
use16
org 100h
    jmp start

n db 4
m db 5

table:
    dw 12,45, 0,82,34
    dw 46,-5,87,11,56
    dw 35,21,77,90,-9
    dw 44,13,-1,99,32
sum rw 4

start:
    movzx cx,[n]
    mov bx,table
    mov di,sum
    xor si,si

rows:
    xor ax,ax
    push cx

    movzx cx,[m]
calc_sum:
    add ax,[bx+si]
    add si,2
    loop calc_sum

    pop cx
    mov [di],ax
    add di,2
    loop rows

    mov ax,4C00h
    int 21h |
```

Рисунок 3 – Код программы

cs:001B F7E3	mul	bx	ax 000B	c=0
cs:001D 8BC8	mov	cx,ax	bx 0012	z=0
cs:001F B80500	mov	ax,0005	cx 00CB	s=0
cs:0022 BB0100	mov	bx,0001	dx 0005	o=0
cs:0025 F7E3	mul	bx	si 0000	p=0
cs:0027 2BC8	sub	cx,ax	di 0000	a=1
cs:0029 B81000	mov	ax,0010	bp 0000	i=1
cs:002C BB0200	mov	bx,0002	sp 0100	d=0
cs:002F 03D8	add	bx,ax	ds 54A4	
cs:0031 8BC1	mov	ax,cx	es 5490	
cs:0033 F7F3	div	bx	ss 54A6	
cs:0035 B44C	mov	ah,4C	cs 54A0	
cs:0037 CD21	int	21	ip 0035	

es:0000 CD 20 FF 9F 00 9A F0 FE = Я bEИ	ss:0102 0000
es:0008 1D F0 E4 01 49 1B AE 01 +Еф@I←o@	ss:0100 0000
es:0010 49 1B 80 02 A4 15 12 07 I←A@nS†*	
es:0018 01 01 01 00 02 FF FF FF @@@ @	

Рисунок 4 – Работа программы Turbo Debugger

После изучения примеров приступил к выполнению задания данного в тексте лабораторной работы, запустил программу в Turbo Debugger и посмотрел её исполнение (Рисунок 5-6).

```

usel6
org 100h
    jmp start

string db '$!olleH'

start:
mov cx,7
xor di,di

lp:
mov al,[string+di]
push ax
inc di
loop lp
xor di,di
mov cx,7
lp_2:
pop ax
mov [string+di],al
inc di
loop lp_2

mov dx,string
mov ah,09h
int 21h
mov ah,08h
int 21h

mov ax,4C00h
int 21h |

```

Рисунок 5 – Код программы

```

cs:0000 B81014 mov ax,1410
cs:0003 8ED8 mov ds,ax
cs:0005 B80000 mov ax,0000
cs:0008 A00000 mov al,[0000]
cs:000B 8A1E0E00 mov bl,[000E]
cs:000F 03C3 add ax,bx
cs:0011 A10F00 mov ax,[000F]
cs:0014 8B1E2100 mov bx,[0021]
cs:0018 B8004C mov ax,4C00
cs:001B CD21 int 21
cs:001D 0000 add [bx+si],al
cs:001F 0012 add [bp+si],dl
cs:0021 0000 add [bx+si],al
cs:0023 0000 add [bx+si],al
cs:0025 006166 add [bx+di+66],ah

ax 4C00 c=0
bx 0034 z=0
cx 0000 s=0
dx 0000 o=0
si 0000 p=1
di 0000 a=0
bp 0000 i=1
sp 0100 d=0
ds 1410
es 13FE
ss 1413
cs 140E
ip 001B

es:0000 CD 20 FB 9F 00 9A F0 FE = √Я бЕ
es:0008 1D F0 32 0B DA 10 0F 07 +E28 r*
es:0010 42 0E 56 01 41 04 25 0E BFW@A%J
es:0018 01 01 01 00 02 04 FF FF @@@ @
es:0020 FF FF FF FF FF FF FF FF

ss:0108 5453
ss:0106 5251
ss:0104 5034
ss:0102 7C34
ss:0100 8634

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

```

Рисунок 6 – Работа программы Turbo Debugger

## Выполнение задания 21:

Перед выполнением задания, изучил теоретический материал в тексте задания (Рисунки 7-8).

### Учебный курс. Часть 21. Простые процедуры

В этой части учебного курса мы рассмотрим основы создания процедур. Процедура представляет собой код, который может выполняться многократно и к которому можно обращаться из разных частей программы. Обычно процедуры предназначены для выполнения каких-то отдельных, законченных действий программы и поэтому их иногда называют подпрограммами. В других языках программирования процедуры могут

называться функциями или методами, но по сути это всё одно и то же

**Команды CALL и RET**

### Рисунок 7 – Изучение материала

#### Ближние и дальние вызовы процедур

Существует 2 типа вызовов процедур. *Ближним* называется вызов процедуры, которая находится в текущем сегменте кода. *Дальний* вызов – это вызов процедуры в другом сегменте. Соответственно существуют 2 вида команды **RET** – для ближнего и дальнего возврата. Компилятор FASM автоматически определяет нужный тип машинной команды, поэтому в большинстве случаев не нужно об этом беспокоиться.

В учебном курсе мы будем использовать только ближние вызовы процедур.

#### Передача параметров

Очень часто возникает необходимость передать процедуре какие-либо параметры. Например, если вы пишете процедуру для вычисления суммы элементов массива, удобно в качестве параметров передавать ей адрес массива и его размер. В таком случае одну и ту же процедуру можно будет использовать для разных массивов в вашей программе. Самый простой способ передать параметры – это поместить их в регистры перед вызовом процедуры.

#### Возвращаемое значение

Кроме передачи параметров часто нужно получить какое-то значение из процедуры. Например, если

процедура что-то вычисляет, хотелось бы получить результат вычисления. А если процедура что-то делает, то полезно узнать, завершилось действие успешно или возникла ошибка. Существуют разные способы возврата значения из процедуры, но самый часто используемый – это поместить значение в один из регистров. Обычно для этой цели используют регистры AL и AX. Хотя вы можете делать так, как вам больше нравится.

### Рисунок 8 – Изучение материала

После изучения материала приступил к рассматриванию примеров программ, запустил программу с помощью Turbo Debugger и посмотрел её исполнение (Рисунок 9-10).

```

usel6
org 100h

    mov ax,myproc
    mov bx,myproc_addr
    xor si,si

    call myproc
    call ax
    call [myproc_addr]
    call word [bx+si]

    mov ax,4C00h
    int 21h

myproc:
    nop
    ret

myproc_addr dw myproc

```

Рисунок 9 – Код программы

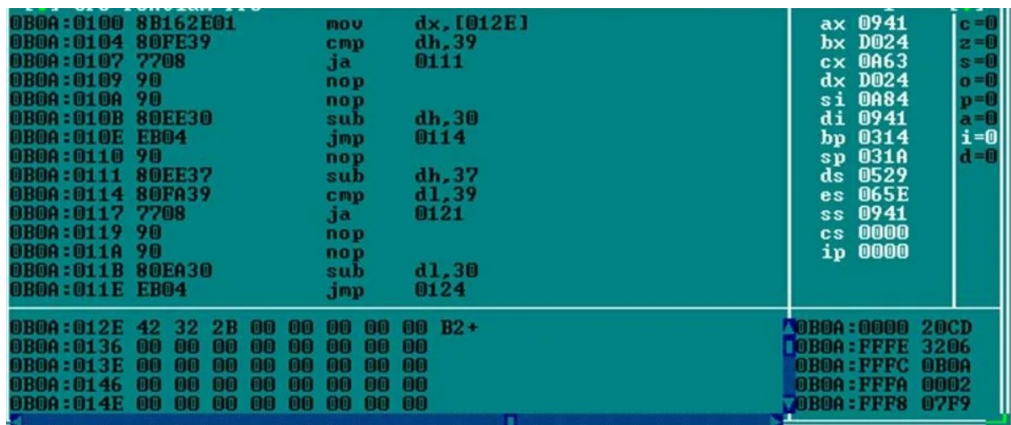


Рисунок 10 – Работа программы Turbo Debugger

После изучения первого примера приступил ко второму, запустил программу в программе Turbo Debugger и посмотрел её исполнение в консоли (Рисунок 11-14).

```
use16
org 100h
    jmp start

msg1 db 'Hello!$'
msg2 db 'asmworld.ru$'
msg3 db 'Press any key...$'

start:
    mov bx,msg1
    call print_message
    mov bx,msg2
    call print_message
    mov bx,msg3
    call print_message

    mov ah,8
    int 21h

    mov ax,4C00h
    int 21h

print_message:
    push ax
    push cx
    push dx

    call get_length
    mov cx,ax
    mov ah,2
    mov dl,0xDA
    int 21h
    mov dl,0xC4
    call draw_line
    mov dl,0xBF
    int 21h
    call print_endline

    mov dl,0xB3
    int 21h
    mov ah,9
    mov dx,bx
    int 21h
    mov ah,2
    mov dl,0xB3
    int 21h
    call print_endline

    mov dl,0xC0
    int 21h
    mov dl,0xC4
    call draw_line
    mov dl,0xD9
    int 21h
    call print_endline

    pop dx
    pop cx
```

Рисунок 11 – Код программы



```

    pop dx
    pop cx
    pop ax
    ret

get_length:
    push bx
    xor ax,ax
str_loop:
    cmp byte[bx],'$'
    je str_end
    inc ax
    inc bx
    jmp str_loop
str_end:
    pop bx
    ret

draw_line:
    push ax
    push cx
    mov ah,2
drl_loop:
    int 21h
    loop drl_loop
    pop cx
    pop ax
    ret

print_endline:
    push ax
    push dx
    mov ah,2
    mov dl,13
    int 21h
    mov dl,10
    int 21h
    pop dx
    pop ax
    ret

```

Рисунок 12 – Код программы

The screenshot displays the Turbo Debugger interface. The main window shows assembly code with the instruction `mov ah,4C` at address `cs:0035` highlighted. To the right, a register window shows the current state of registers: `ax 000B`, `bx 0012`, `cx 00CB`, `dx 0005`, `si 0000`, `di 0000`, `bp 0000`, `sp 0100`, `ds 54A4`, `es 5490`, `ss 54A6`, `cs 54A0`, and `ip 0035`. A status bar at the bottom right shows `ss:0102 0000` and `ss:0100 0000`. The bottom of the window shows a hex dump of memory starting at `es:0000`.

Рисунок 13 – Работа программы Turbo Debugger



Рисунок 14 – Вывод в консоли