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.

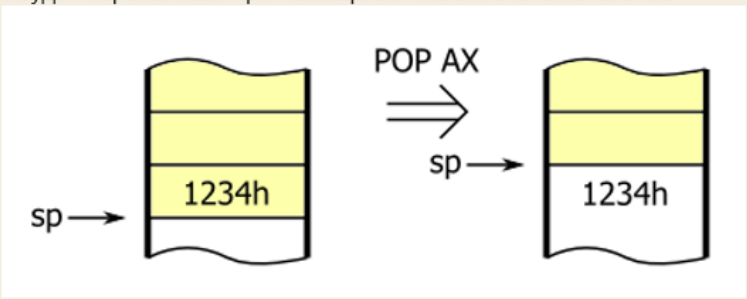
Рисунок 1 – Изучение материала

Извлечение элемента из стека

Выполняется командой POP. У этой команды также один операнд, который может быть 16-битным регистром (в том числе сегментым, 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

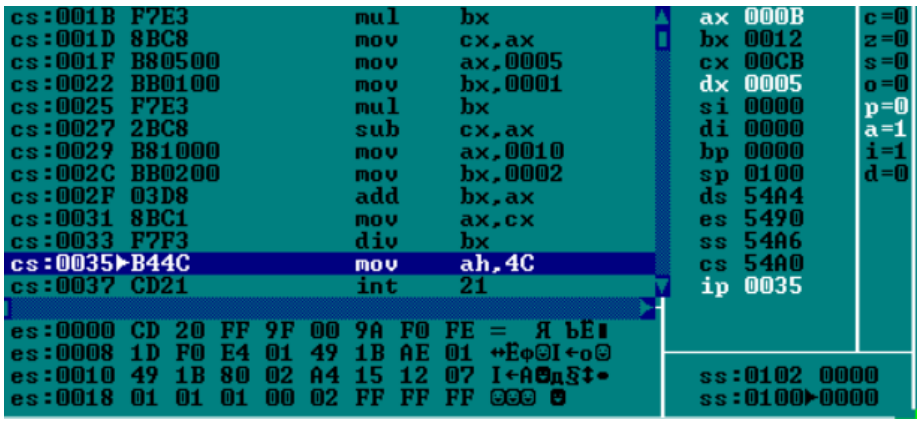


Рисунок 4 – Работа программы Turbo Debugger

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

```
use16
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

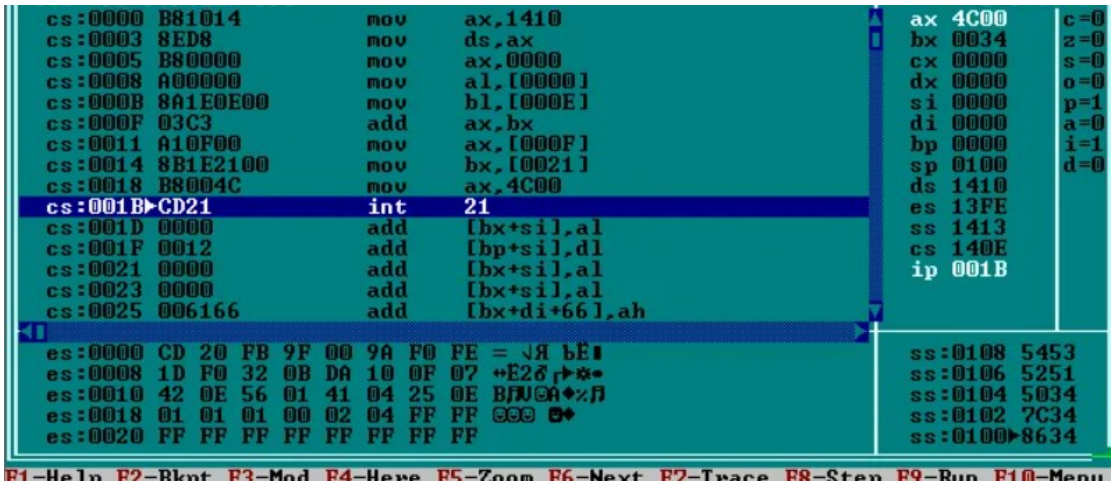


Рисунок 6 – Работа программы Turbo Debugger

Выполнение задания 21:

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

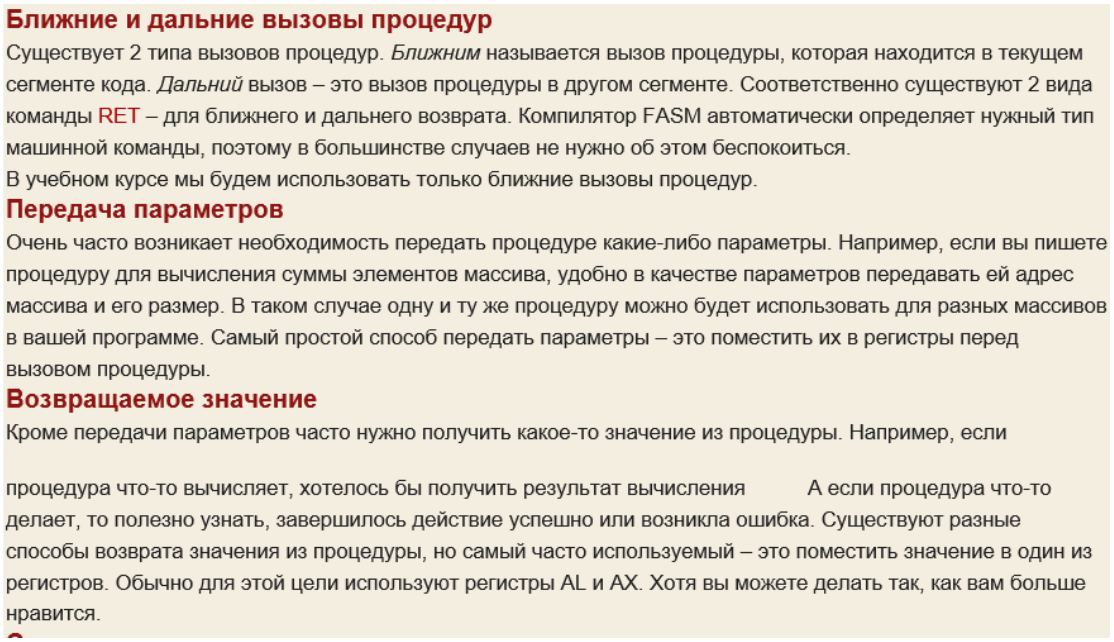
Учебный курс. Часть 21. Простые процедуры

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).

```
use16
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

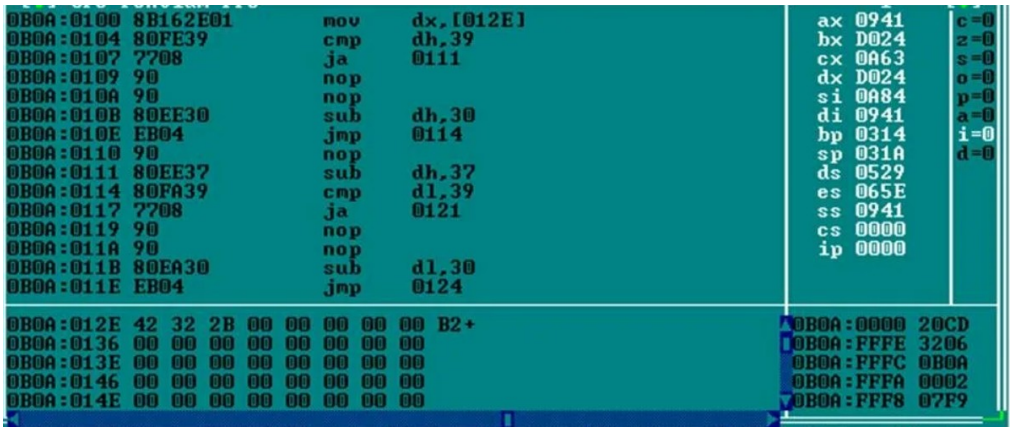


Рисунок 10 – Работа программы Turbo Debugger

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

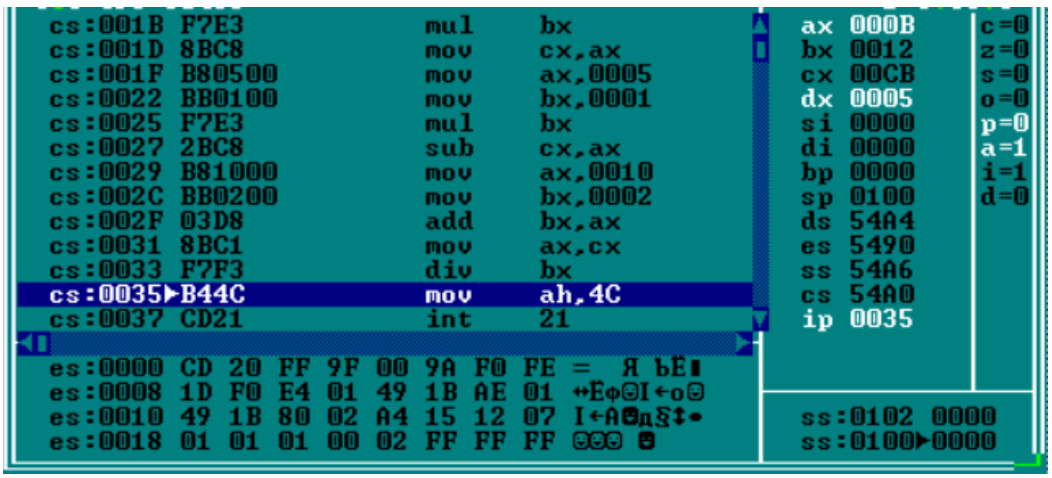


Рисунок 13 – Работа программы Turbo Debugger

Рисунок 14 – Вывод в консоли