

# **Лабораторная работа №8**

**Архитектура компьютера**

Иванов Александр

# Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	7
4	Самостоятельная работа	13
5	Выводы	14
	Список литературы	15

## Список иллюстраций

3.1	использование команды ‘mkdir’ и ‘touch’ . . . . .	7
3.2	окно MidnightCommander . . . . .	8
3.3	результат выполнения кода . . . . .	8
3.4	окно MidnightCommander . . . . .	8
3.5	результат выполнения кода . . . . .	9
3.6	окно MidnightCommander . . . . .	9
3.7	результат выполнения кода . . . . .	9
3.8	окно MidnightCommander . . . . .	10
3.9	результат выполнения кода . . . . .	10
3.10	окно MidnightCommander . . . . .	11
3.11	результат выполнения кода . . . . .	11
3.12	окно MidnightCommander . . . . .	12
3.13	результат выполнения кода . . . . .	12
4.1	окно MidnightCommander . . . . .	13
4.2	результат выполнения кода . . . . .	13

## Список таблиц

# 1 Цель работы

Приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки.

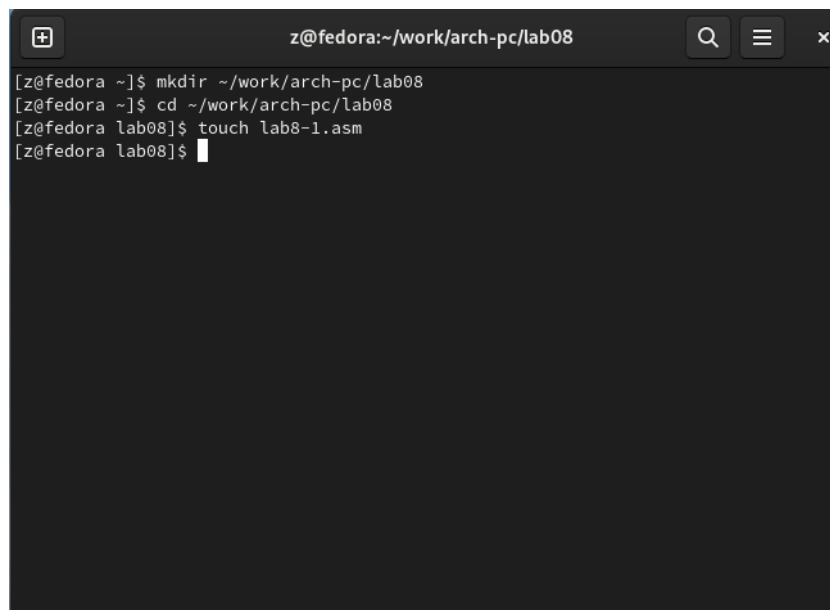
## 2 Теоретическое введение

Стек – это структура данных, организованная по принципу LIFO («Last In – First Out»

или «последним пришёл – первым ушёл»). Стек является частью архитектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды. Основной функцией стека является функция сохранения адресов возврата и передачи аргументов при вызове процедур. Кроме того, в нём выделяется память для локальных переменных и могут временно храниться значения регистров. На рис. 8.1 показана схема организации стека в процессоре. Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается. |

### 3 Выполнение лабораторной работы

Создание рабочего каталога и рабочего файла (рис. 3.1).

A terminal window titled 'z@fedora:~/work/arch-pc/lab08' with search, menu, and close buttons. The terminal shows the following commands and output:

```
[z@fedora ~]$ mkdir ~/work/arch-pc/lab08
[z@fedora ~]$ cd ~/work/arch-pc/lab08
[z@fedora lab08]$ touch lab8-1.asm
[z@fedora lab08]$
```

Рис. 3.1: использование команды 'mkdir' и 'touch'

Написание кода вывода чисел меньше переменной(рис. 3.2).

```

; ---- Преобразование 'N' из символа в число
mov eax,N
call atoi
mov [N],eax
; ----- Организация цикла
mov ecx,[N] ; Счетчик цикла, `ecx=N`
label:
sub ecx,1 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF
call quit

```

<sup>^G</sup> Справка    <sup>^O</sup> Записать    <sup>^W</sup> Поиск    <sup>^K</sup> Вырезать    <sup>^T</sup> Выполнить    <sup>^C</sup> Позиция  
<sup>^X</sup> Выход    <sup>^R</sup> ЧитФайл    <sup>^\_</sup> Замена    <sup>^U</sup> Вставить    <sup>^J</sup> Выводить    <sup>^/</sup> К строке

Рис. 3.2: окно MidnightCommander

Создание и запуск программы(рис. 3.3).

```

[z@fedora lab08]$ nasm -f elf lab8-1.asm
[z@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[z@fedora lab08]$ ./lab8-1
Введите N: 12
12
11
10
9
8
7
6
5
4
3
2
1
[z@fedora lab08]$

```

Рис. 3.3: результат выполнения кода

Редактирование кода вывода числа меньше переменной на 1(рис. 3.4).

```

label:
push ecx ; добавление значения ecx в стек
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF
pop ecx ; извлечение значения ecx из стека
loop label
call quit

```

Рис. 3.4: окно MidnightCommander

Создание и запуск программы(рис. 3.5).



```
[z@fedora lab08]$ nasm -f elf lab8-1.asm
[z@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[z@fedora lab08]$ ./lab8-1
Введите N: 12
11
[z@fedora lab08]$
```

Рис. 3.5: результат выполнения кода

Редактирование кода вывода чисел меньше переменной,(рис. 3.6).

```
label:
push ecx ; добавление значения ecx в стек
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF
pop ecx ; извлечение значения ecx из стека
loop label
call quit
```

Рис. 3.6: окно MidnightCommander

Создание и запуск программы(рис. 3.7).

```
ld -m elf_i386 -o nasm -f elf lab8-1.asm
[z@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[z@fedora lab08]$ ./lab8-1
Введите N: 12
11
10
9
8
7
6
5
4
3
2
1
0
```

Рис. 3.7: результат выполнения кода

Написание кода обработки аргументов командной строки(рис. 3.8).

```

SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
next:
cmp ecx, 0 ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем аргумент из стека
call printf ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit

```

Рис. 3.8: окно MidnightCommander

Создание и запуск программы(рис. 3.9).

```

[z@fedora lab08]$ nasm -f elf lab8-1.asm
[z@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[z@fedora lab08]$ ./lab8-1 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3

```

Рис. 3.9: результат выполнения кода

Написание кода для вычисления суммы чисел меньше переменной(рис. 3.10).

```
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
```

Сохранить изменённый буфер?

☐ Да

☒ Нет

Рис. 3.10: окно MidnightCommander

Создание и запуск программы(рис. 3.11).

```
[z@fedora lab08]$ ./lab8-1 12 13 7 10 5
Результат: 47
[z@fedora lab08]$
```

Рис. 3.11: результат выполнения кода

Написание кода для вычисления произведения чисел меньше переменной(рис. 3.12).

```

pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
mov esi, 1 ; Используем `esi` для хранения
; промежуточных сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
mul esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint

```

Рис. 3.12: окно MidnightCommander

Создание и запуск программы(рис. 3.13).

```

[z@fedora lab08]$ nasm -f elf lab8-1.asm
[z@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[z@fedora lab08]$ ./lab8-1 12 13 7 10 5
результат: 54600
[z@fedora lab08]$

```

Рис. 3.13: результат выполнения кода

## 4 Самостоятельная работа

Написание кода для вычисления суммы результатов функции(рис. 4.1).

```
%include 'in_out.asm'

SECTION .data
f_x db "функция: 10x-4",0h
msg db 10,13,'результат: ',0h

SECTION .text
global _start

_start:
pop ecx
pop edx
mul ecx, 10
mov esi, 0

next:
cmp ecx,0h
jz _end
pop eax
call atoi
```

Рис. 4.1: окно MidnightCommander

Создание и запуск программы(рис. 4.2).

```
[z@fedora lab08]$ ./main 1 2 3 4
Функция: f(x)=10x-4
Результат: 84
[z@fedora lab08]$
```

Рис. 4.2: результат выполнения кода

## 5 Выводы

В ходе выполнения лабораторной работы я приобрел навыки написания программ с использованием циклов и обработкой аргументов командной строки

## **Список литературы**