

Отчет по лабораторной работе №8

Дисциплина: архитектура компьютера

Абронина Алиса Кирилловна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Задание для самостоятельной работы	14
6	Выводы	16
	Список литературы	17

Список иллюстраций

4.1	Создание каталога	8
4.2	Копирование программы из листинга	8
4.3	Запуск программы	9
4.4	Изменение программы	9
4.5	Запуск измененной программы	10
4.6	Добавление push и pop в цикл программы	10
4.7	Запуск измененной программы	10
4.8	Копирование программы из листинга	11
4.9	Запуск второй программы	11
4.10	Копирование программы из третьего листинга	12
4.11	Запуск третьей программы	12
4.12	Изменение третьей программы	13
4.13	Запуск измененной третьей программы	13
5.1	Написание программы для самостоятельной работы	14
5.2	Запуск программы для самостоятельной работы	15

Список таблиц

1 Цель работы

Освоение программирования с применением циклов и обработкой аргументов командной строки.

2 Задание

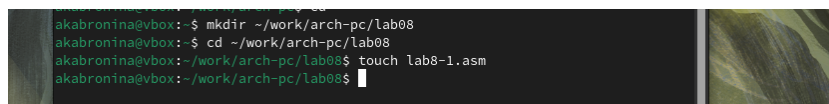
1. Выполнение лабораторной работы
2. Выполнение заданий для самостоятельной работы

3 Теоретическое введение

Стек — это место для хранения данных, где последним добавленное всегда извлекается первым. Он встроен прямо в процессор компьютера и управляется специальными командами и регистрами. Основная работа стека заключается в том, чтобы запоминать адреса, куда нужно вернуться после выполнения процедуры, а также передавать данные между ними. В стеке также можно хранить временные переменные и значения регистров.

4 Выполнение лабораторной работы

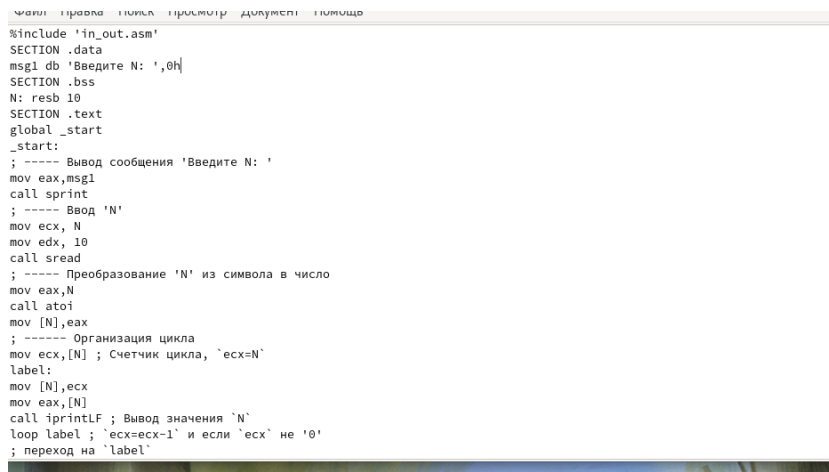
Создаю каталог для программ лабораторной работы №8 (рис. 4.1).



```
akabronina@vbox: ~$ mkdir ~/work/arch-pc/lab08
akabronina@vbox: ~$ cd ~/work/arch-pc/lab08
akabronina@vbox: ~/work/arch-pc/lab08$ touch lab8-1.asm
akabronina@vbox: ~/work/arch-pc/lab08$
```

Рис. 4.1: Создание каталога

Копирую в созданный файл программу из листинга (рис. 4.2).



```
%include 'in_out.asm'
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:
; ----- Вывод сообщения 'Введите N: '
mov eax,msg1
call sprint
; ----- Ввод 'N'
mov ecx, N
mov edx, 10
call sread
; ----- Преобразование 'N' из символа в число
mov eax,N
call atoi
mov [N],eax
; ----- Организация цикла
mov ecx,[N] ; Счетчик цикла, 'ecx=N'
label:
mov [N],ecx
mov eax,[N]
call iprintf ; Вывод значения 'N'
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
```

Рис. 4.2: Копирование программы из листинга

Запускаю программу, она показывает работу циклов в NASM (рис. 4.3).


```
akabronina@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
akabronina@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
akabronina@vbox:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 9
9
8
7
6
5
4
3
2
1
akabronina@vbox:~/work/arch-pc/lab08$
```

Рис. 4.3: Запуск программы

Заменяю программу изначальную так, что в теле цикла я изменяю значение регистра `ecx` (рис. 4.4).

```
*~/work/arch-pc/lab08/lab8-1.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь

#include 'in_out.asm'

SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:

mov eax,msg1
call sprint

mov ecx, N
mov edx, 10
call sread

mov eax,N
call atoi

mov [N], eax
mov ecx, [N]
label:
sub ecx, 1
mov [N], ecx
mov eax, [N]
call iprintLF
loop label
call quit
```

Рис. 4.4: Изменение программы

Из-за того, что теперь регистр `ecx` на каждой итерации уменьшается на 2 значения, количество итераций уменьшается вдвое (рис. 4.5).

```
akabronina@vbox:~/work/arch-pc/lab08$ mousepad lab8-1.asm
akabronina@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
akabronina@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
akabronina@vbox:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
7
5
3
1
akabronina@vbox:~/work/arch-pc/lab08$
```

Рис. 4.5: Запуск измененной программы

Добавляю команды push и pop в программу (рис. 4.6).

```
Файл  Правка  Поиск  Просмотр  Документ  Помощь
%include 'in_out.asm'

SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:

mov eax,msg1
call sprint

mov ecx, N
mov edx, 10
call sread

mov eax,N
call atoi

mov [N], eax
mov ecx, [N]
label:
push ecx
sub ecx, 1
mov [N], ecx
mov eax, [N]
call iprintLF
pop ecx
loop label
call quit
```

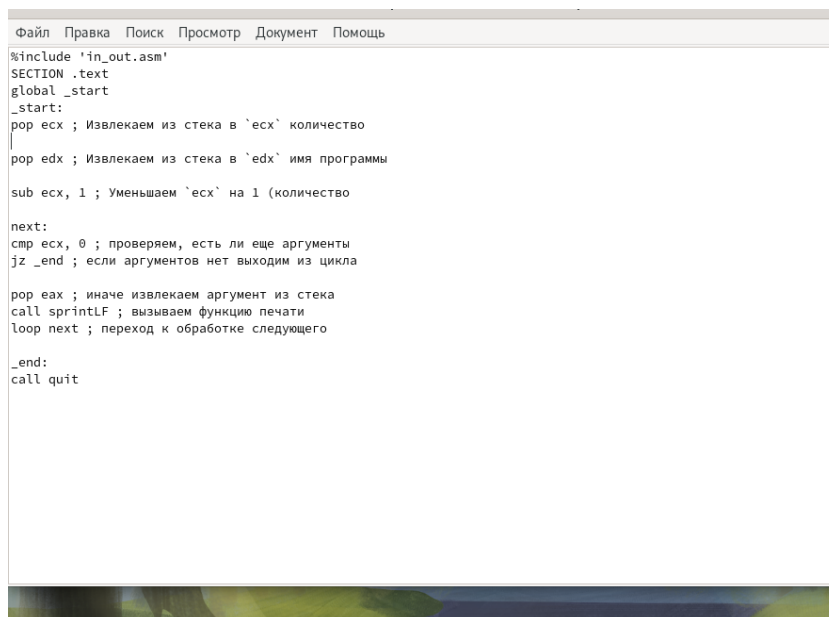
Рис. 4.6: Добавление push и pop в цикл программы

Теперь количество итераций совпадает введенному N, но произошло смещение выводимых чисел на 1 (рис. 4.7).

```
akabronina@vbox:~/work/arch-pc/lab08$ mousepad lab8-1.asm
akabronina@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
akabronina@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
akabronina@vbox:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
akabronina@vbox:~/work/arch-pc/lab08$ touch lab8-2.asm
```

Рис. 4.7: Запуск измененной программы

Создаю новый файл для программы и копирую в него код из следующего листинга (рис. 4.8).



```
Файл  Правка  Поиск  Просмотр  Документ  Помощь
#include 'in_out.asm'
SECTION .text
global _start
_start:
    pop ecx ; Извлекаем из стека в 'ecx' количество
    pop edx ; Извлекаем из стека в 'edx' имя программы

    sub ecx, 1 ; Уменьшаем 'ecx' на 1 (количество)

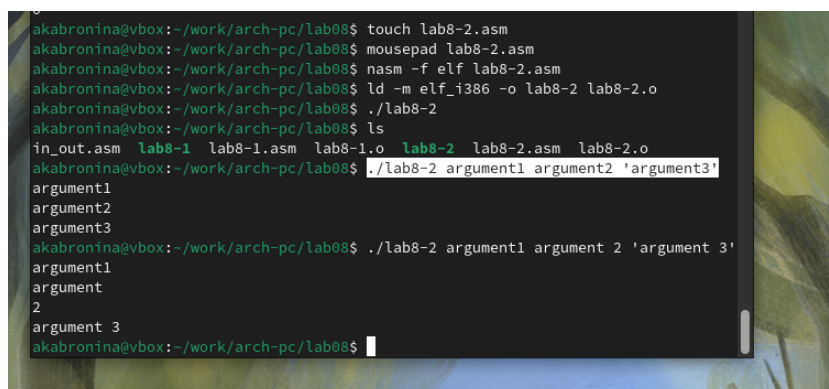
next:
    cmp ecx, 0 ; проверяем, есть ли еще аргументы
    jz _end ; если аргументов нет выходим из цикла

    pop eax ; иначе извлекаем аргумент из стека
    call sprintf ; вызываем функцию печати
    loop next ; переход к обработке следующего

_end:
    call quit
```

Рис. 4.8: Копирование программы из листинга

Компилирую программу и запускаю, указав аргументы. Программой было обработано то же количество аргументов, что и было введено (рис. 4.9).



```
akabronina@vbox:~/work/arch-pc/lab08$ touch lab8-2.asm
akabronina@vbox:~/work/arch-pc/lab08$ mousepad lab8-2.asm
akabronina@vbox:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
akabronina@vbox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
akabronina@vbox:~/work/arch-pc/lab08$ ./lab8-2
akabronina@vbox:~/work/arch-pc/lab08$ ls
in_out.asm  lab8-1  lab8-1.asm  lab8-1.o  lab8-2  lab8-2.asm  lab8-2.o
akabronina@vbox:~/work/arch-pc/lab08$ ./lab8-2 argument1 argument2 'argument3'
argument1
argument2
argument3
akabronina@vbox:~/work/arch-pc/lab08$ ./lab8-2 argument1 argument 2 'argument 3'
argument1
argument
2
argument 3
akabronina@vbox:~/work/arch-pc/lab08$
```

Рис. 4.9: Запуск второй программы

Создаю новый файл для программы и копирую в него код из третьего листинга (рис. 4.10).

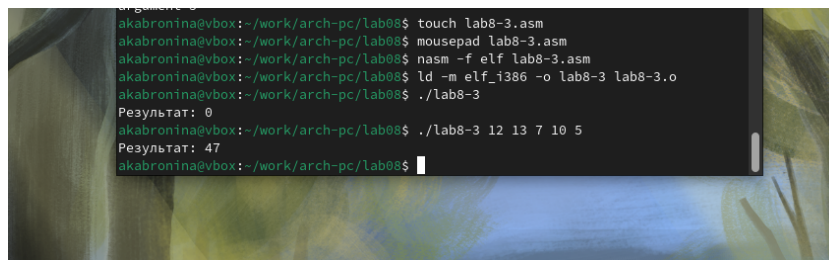
```

%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
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)
    mov esi, 0 ; 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
    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 ;

```

Рис. 4.10: Копирование программы из третьего листинга

Компилирую программу и запускаю, указав в качестве аргументов некоторые числа, программа их складывает (рис. 4.11).



```

akabronina@vbox: ~/work/arch-pc/lab08$ touch lab8-3.asm
akabronina@vbox: ~/work/arch-pc/lab08$ mousepad lab8-3.asm
akabronina@vbox: ~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
akabronina@vbox: ~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
akabronina@vbox: ~/work/arch-pc/lab08$ ./lab8-3
Результат: 0
akabronina@vbox: ~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
akabronina@vbox: ~/work/arch-pc/lab08$

```

Рис. 4.11: Запуск третьей программы

Изменяю поведение программы так, чтобы указанные аргументы она умножала, а не складывала (рис. 4.12).

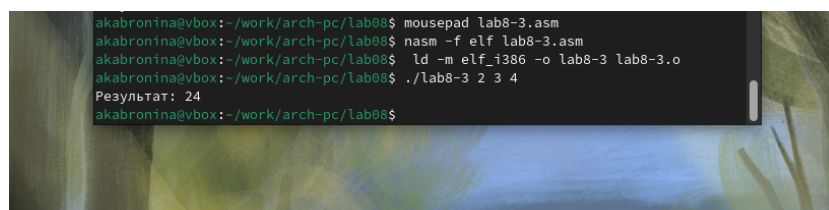
```
Файл  Правка  Поиск  Просмотр  Документ  Помощь

%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
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)
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
mov 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 ;
```

Рис. 4.12: Изменение третьей программы

Программа теперь умножает данные на вход числа (рис. 4.13).




```
akabronina@vbox: ~/work/arch-pc/lab08$ mousepad lab8-3.asm
akabronina@vbox: ~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
akabronina@vbox: ~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
akabronina@vbox: ~/work/arch-pc/lab08$ ./lab8-3 2 3 4
Результат: 24
akabronina@vbox: ~/work/arch-pc/lab08$
```

Рис. 4.13: Запуск измененной третьей программы

5 Задание для самостоятельной работы

Пишу программу, которая будет находить сумма значений для функции $f(x) = 17+5x$, которая совпадает с моим 18 вариантом (рис. 5.1).



```
Файлы  |  правка  |  поиск  |  просмотр  |  документ  |  помощь
%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 17 + 5x", 0
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_start:
mov eax, msg_func
call sprintf

pop ecx
pop edx
sub ecx, 1
mov esi, 0

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

mov ebx, 5
mul ebx
add eax, 17

add esi, eax
loop next
```

Рис. 5.1: Написание программы для самостоятельной работы

Проверяю работу программы, указав в качестве аргумента несколько чисел(рис. 5.2).

```
Результат: 24
akabronina@vbox: ~/work/arch-pc/lab08$ touch lab8-sr.asm
akabronina@vbox: ~/work/arch-pc/lab08$ mousepad lab8-sr.asm
akabronina@vbox: ~/work/arch-pc/lab08$ nasm -f elf lab8-sr.asm
akabronina@vbox: ~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-sr lab8-sr.o
akabronina@vbox: ~/work/arch-pc/lab08$ ./lab8-sr
Функция:  $f(x) = 17 + 5x$ 
Результат: 0
akabronina@vbox: ~/work/arch-pc/lab08$ ./lab8-sr 1 2 3
Функция:  $f(x) = 17 + 5x$ 
Результат: 81
akabronina@vbox: ~/work/arch-pc/lab08$ mousepad lab8-sr.asm
akabronina@vbox: ~/work/arch-pc/lab08$
```

Рис. 5.2: Запуск программы для самостоятельной работы

6 Выводы

В результате выполнения данной лабораторной работы я приобрела навыки написания программ с использованием циклов, а также научилась обрабатывать аргументы командной строки.

Список литературы