

Отчёт по лабораторной работе №8

Дисциплина: Архитектура компьютеров и операционные системы

Чистов Даниил Максимович

Содержание

1	Цель работы	4
2	Задание	5
3	Выполнение лабораторной работы	6
3.1	Реализация циклов в NASM	6
3.2	Обработка аргументов командной строки	11
3.3	Задание для самостоятельной работы	16
4	Выводы	20
	Список литературы	21

Список иллюстраций

3.1	Подготовка к выполнению заданий	6
3.2	Код программы листинга 8.1	7
3.3	Успешно рабочая программа	8
3.4	Новый код программы	8
3.5	Программа работает некорректно	9
3.6	Изменённый код программы	10
3.7	Программа работает корректно	11
3.8	Создание файла	11
3.9	Код программы lab8-2	12
3.10	Успешная работа программы	13
3.11	Создание файла	13
3.12	Код программы lab8-3	14
3.13	Проверка работы программы	15
3.14	Новый код программы для произведения	15
3.15	Программа работает успешно	16
3.16	Код программы для задания	17
3.17	Программа работает успешно!	19

1 Цель работы

Цель данной работы - приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки.

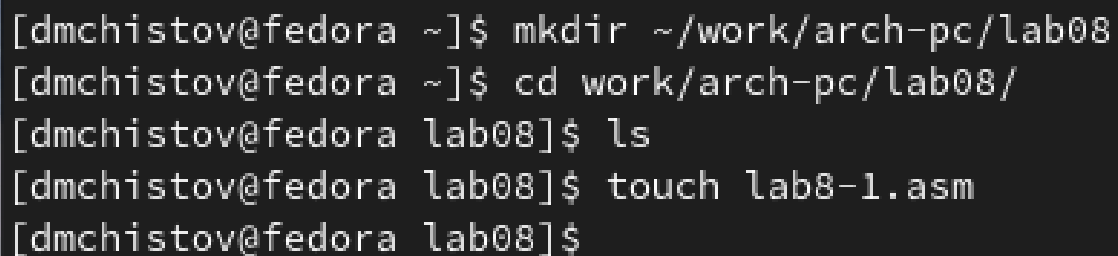
2 Задание

- Реализация циклов в NASM
- Обработка аргументов командной строки
- Задание для самостоятельной работы

3 Выполнение лабораторной работы

3.1 Реализация циклов в NASM

Создаю каталог для программ лабораторной работы №8, затем создаю файл lab8-1.asm (рис. 3.1).

A terminal window with a dark background and light gray text. The prompt is [dmchistov@fedora ~]. The user enters 'mkdir ~/work/arch-pc/lab08'. The prompt changes to [dmchistov@fedora ~]\$. The user enters 'cd work/arch-pc/lab08/'. The prompt changes to [dmchistov@fedora lab08]\$. The user enters 'ls'. The prompt changes to [dmchistov@fedora lab08]\$. The user enters 'touch lab8-1.asm'. The prompt changes to [dmchistov@fedora lab08]\$.

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

Рис. 3.1: Подготовка к выполнению заданий

Открываю созданный мной файл в mcedit и вставляю код из листинга 8.1 (рис. 3.2).

```

lab8-1.asm      [----]  0 L:[  1+ 0  1/ 36] *(0
;-----
; Программа вывода значений регистра 'ecx'
;-----
#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 iprintLF ; Вывод значения `N`
loop label ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`
call quit

```

Рис. 3.2: Код программы листинга 8.1

Создаю исполняемый файл и проверяю его работу (рис. 3.3).

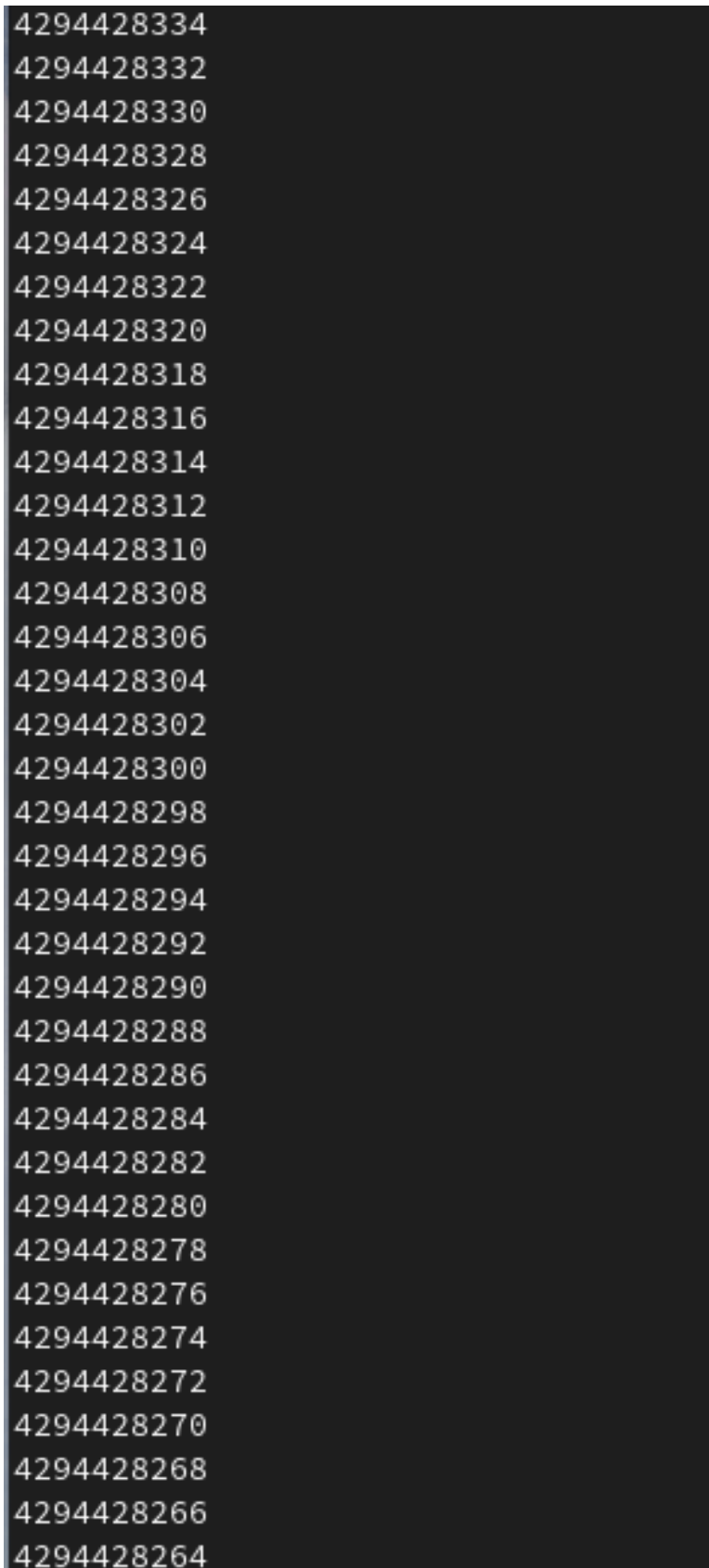
```
[dmchistov@fedora lab08]$ nasm -f elf lab8-1.asm
[dmchistov@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[dmchistov@fedora lab08]$ ./lab8-1
Введите N: 5
5
4
3
2
1
[dmchistov@fedora lab08]$
```

Рис. 3.3: Успешно рабочая программа

Изменяю код, воспользовавшись кодом, приведённом в задании (рис. 3.4), затем проверяю его работу (рис. 3.5).

```
; ----- Организация цикла
mov ecx, [N] ; Счетчик цикла, `ecx=N`
label:
sub ecx, 1 ; `ecx=ecx-1`
mov [N], ecx
mov eax, [N]
call iprintLF
loop label.
```

Рис. 3.4: Новый код программы



4294428334
4294428332
4294428330
4294428328
4294428326
4294428324
4294428322
4294428320
4294428318
4294428316
4294428314
4294428312
4294428310
4294428308
4294428306
4294428304
4294428302
4294428300
4294428298
4294428296
4294428294
4294428292
4294428290
4294428288
4294428286
4294428284
4294428282
4294428280
4294428278
4294428276
4294428274
4294428272
4294428270
4294428268
4294428266
4294428264

Рис. 3.5: Программа работает некорректно

Данный пример показал, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы. Программа просто выводит бесконечный набор чисел.

Теперь попытаюсь корректно использовать регистр `ecx`, воспользовавшись предоставленным заданием кодом. (рис. 3.6) и также проверяю корректность работы программы (рис. 3.7).

```
label:
mov ecx,[N]
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
; переход на `label`
call quit
```

Рис. 3.6: Изменённый код программы

```
[dmchistov@fedora lab08]$ nasm -f elf lab8-1.asm
[dmchistov@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[dmchistov@fedora lab08]$ ./lab8-1
Введите N: 5
4
3
2
1
0
[dmchistov@fedora lab08]$
```

Рис. 3.7: Программа работает корректно

Успешно! В данном случае количество выводов соответствует с введённым числом N.

3.2 Обработка аргументов командной строки

Внимательно изучаю код из листинга 8.2, после чего создаю файл lab8-2.asm (рис. 3.8), вставляю код (рис. 3.9), создаю файл и указываю следующие аргументы: аргумент1 аргумент 2 'аргумент 3' (рис. 3.10)

```
[dmchistov@fedora lab08]$ touch lab8-2.asm
[dmchistov@fedora lab08]$
```

Рис. 3.8: Создание файла

```

;-----
; Обработка аргументов командной строки
;-----
#include 'in_out.asm'
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 sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit

```

Рис. 3.9: Код программы lab8-2

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

Рис. 3.10: Успешная работа программы

Программа работает успешно!

Теперь создаю новый файл - lab8-3.asm (рис. 3.11), и ввожу в него код из листинга 8.3 (рис. 3.12).

```
аргумент 3
[dmchistov@fedora lab08]$ touch lab8-3.asm
```

Рис. 3.11: Создание файла

```

%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 ; завершение программы

```

Рис. 3.12: Код программы lab8-3

Создаю файл и указываю аргументы такие же, как в приведённом примере: 12 13 7 10 5 (рис. 3.13).

```
[dmchistov@fedora lab08]$ nasm -f elf lab8-3.asm
[dmchistov@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[dmchistov@fedora lab08]$ ./lab8-3 12 13 7 10 5
Результат: 47
[dmchistov@fedora lab08]$
```

Рис. 3.13: Проверка работы программы

Программа работает успешно!

Теперь от меня требуется изменить код программы 8-3 таким образом, чтобы вместо суммирования аргументов было выполнено произведение. Для этого воспользуюсь обозначением `mul`, а также перед этим изменю значение `esi` с 0 на 1, таким образом программа будет умножать каждый аргумент на следующий. Приведу изменённый код программы (рис. 3.14).

```
<----->; 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 ; добавляем к промежуточной сумме
<----->; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
```

Рис. 3.14: Новый код программы для произведения

Проверяю корректность работы программы (рис. 3.15).

```
[dmchistov@fedora lab08]$ nasm -f elf lab8-3.asm
[dmchistov@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[dmchistov@fedora lab08]$ ./lab8-3 5 4 5
Результат: 100
```

Рис. 3.15: Программа работает успешно

3.3 Задание для самостоятельной работы

От меня требуется написать программу, которая суммирует значения определённой функции $f(x)$. Так как мой вариант - 2, моя функция выглядит следующим образом:

$$f(x) = 3x - 1$$

Вот готовый код программы в виде скриншота (рис. 3.16).


```

%include 'in_out.asm'
SECTION .data
msg1 db "Функция: f(x)=3x-1 ", 0x0A
msg2 db "Результат: "
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
mov ebx, 3 ; Храним 3, как один из операндов в ф-ии f(x)
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 eax
mov ebx, eax ; умножаем x на 3
sub ebx, 1 ; вычитаем 1
add esi, ebx ; добавляем готовый результат к общей сумме
loop next ; переход к обработке следующего аргумента

_end:
mov eax, msg1.
call sprint

mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы

```

Рис. 3.16: Код программы для задания

Вот код в письменном виде:

```
%include 'in_out.asm'

SECTION .data
msg1 db "Функция: f(x)=3x-1", 0x0A
msg2 db "Результат:"

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

mov ebx, 3 ; Храним 3, как один из операндов в ф-ии f(x)
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла

; (переход на метку `_end` )

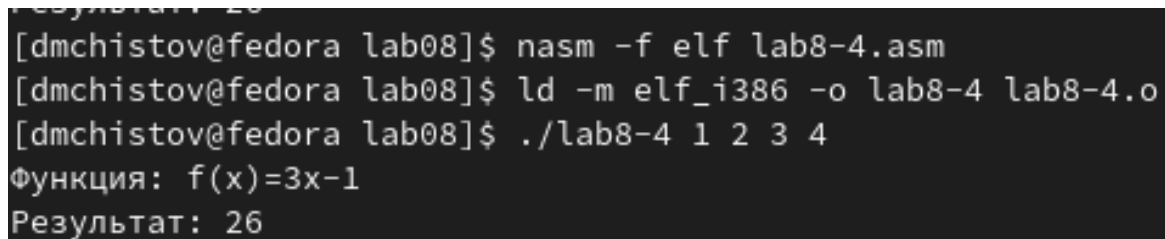
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
mul eax
```

```

mov ebx, eax ; умножаем x на 3
sub ebx, 1 ; вычитаем 1
add esi, ebx ; добавляем готовый результат к общей сумме
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg1
call sprint
mov eax, esi ; записываем сумму в регистр eax
call iprintLF ; печать результата
call quit ; завершение программы

```

Теперь нужно проверить корректность работы моего кода (рис. 3.17).



```

[dmchistov@fedora lab08]$ nasm -f elf lab8-4.asm
[dmchistov@fedora lab08]$ ld -m elf_i386 -o lab8-4 lab8-4.o
[dmchistov@fedora lab08]$ ./lab8-4 1 2 3 4
Функция: f(x)=3x-1
Результат: 26

```

Рис. 3.17: Программа работает успешно!

4 Выводы

Я выполнил поставленные цели лабораторной работы, благодаря этому я приобрёл навыки написания программ с использованием циклов и обработкой аргументов командной строки.

Список литературы

Лабораторная работы №8