

Отчёт по лабораторной работе №8

дисциплина: Архитектура компьютера

Аносов Даниил Игоревич

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
3.1	Реализация циклов в NASM	7
3.2	Обработка аргументов командной строки	13
3.2.1	Вычисление суммы аргументов	15
3.2.2	Вычисление произведения аргументов	16
4	Задание для самостоятельной работы	18
5	Выводы	22

Список иллюстраций

3.1	Создание каталога для программ	7
3.2	Открытый Vim	8
3.3	Компиляция и первый запуск программы	9
3.4	Vim с обновленной программой	10
3.5	Повторная компиляция и запуск программы	10
3.6	Vim с обновленной программой	12
3.7	Компиляция и запуск новой программы	13
3.8	Редактирование файла <i>lab8-2.asm</i>	14
3.9	Повторная компиляция и запуск новой программы	14
3.10	Vim с файлом <i>lab8-3.asm</i>	15
3.11	Компиляция и запуск <i>lab8-3.asm</i>	16
3.12	Vim с измененным файлом <i>lab8-3.asm</i>	17
3.13	Компиляция и запуск <i>lab8-3.asm</i>	17
4.1	Vim с файлом <i>task.asm</i>	19
4.2	Проверка работы <i>task.asm</i>	19
4.3	Vim с обновленным файлом <i>task.asm</i>	20
4.4	Тестирование программы <i>task.asm</i>	21
4.5	Загрузка файлов на GitHub	21

Список таблиц

1 Цель работы

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

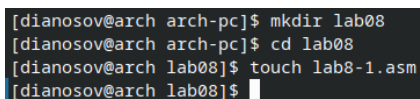
2 Задание

1. Напишите программу, которая находит сумму значений функции $f(x)$ для $x = x_1, x_2, \dots, x_n$, т.е. программа должна выводить значение $f(x_1) + f(x_2) + \dots + f(x_n)$. Значения x_i передаются как аргументы. Вид функции $f(x)$ выбрать из таблицы 8.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Создайте исполняемый файл и проверьте его работу на нескольких наборах $x = x_1, x_2, \dots, x_n$.

3 Выполнение лабораторной работы

3.1 Реализация циклов в NASM

Откроем терминал и создадим каталог для программ лабораторной работы №8. В новом каталоге создадим файл для первой программы *lab8-1.asm*. (рис. 3.1).



```
[dianosov@arch arch-pc]$ mkdir lab08  
[dianosov@arch arch-pc]$ cd lab08  
[dianosov@arch lab08]$ touch lab8-1.asm  
[dianosov@arch lab08]$
```

Рис. 3.1: Создание каталога для программ

Введём в этот файл текст программы из предложенного листинга. (рис. 3.2).


```

[dianosov@arch lab08]$ cp ../lab07/in_out.asm .
[dianosov@arch lab08]$ ls
in_out.asm  lab8-1.asm
[dianosov@arch lab08]$ nasm -f elf lab8-1.asm
[dianosov@arch lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[dianosov@arch lab08]$ ./lab8-1
Введите N: 3
3
2
1
[dianosov@arch lab08]$

```

Рис. 3.3: Компиляция и первый запуск программы

Изменим текст программы, добавив изменение значения регистра `ecx` в цикле (рис. 3.4).

```

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

```


Какие значения принимает регистр `ecx` в цикле? Соответствует ли число проходов цикла значению N , введенному с клавиатуры?

Регистр `ecx` принимает значения с шагом 2, начиная с $N - 1$. Поэтому, число проходов цикла равно $\lfloor \frac{N}{2} \rfloor$, что меньше N .

Для использования регистра `ecx` в цикле и сохранения корректности работы программы можно использовать стек. Внесём изменения в текст программы, добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop`:

```
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
```

Откроем файл программы в **Vim** и отредактируем код (рис. 3.6).


```

[dianosov@arch lab08]$ nasm -f elf lab8-1.asm
[dianosov@arch lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[dianosov@arch lab08]$ ./lab8-1
Введите N: 8
7
6
5
4
3
2
1
0
[dianosov@arch lab08]$ ./lab8-1
Введите N: 5
4
3
2
1
0
[dianosov@arch lab08]$

```

Рис. 3.7: Компиляция и запуск новой программы

Теперь, после того, как мы начали сохранять значения итератора `ecx` в стек, цикл стал работать корректно. Количество проходов цикла соответствует введённому N .

3.2 Обработка аргументов командной строки

При разработке программ иногда встает необходимость указывать аргументы, которые будут использоваться в программе, непосредственно из командной строки при запуске программы. В качестве примера такой программы рассмотрим программу, предложенную в листинге. Создадим для неё новый файл `lab8-2.asm` и откроем его в редакторе **Vim**.

Программа обработала не 3 аргумента, а 4, так как слова, разделённые пробелом, написанные без кавычек (аргумент 2) обрабатываются как отдельные.

3.2.1 Вычисление суммы аргументов

Теперь создадим новый файл под названием *lab8-3.asm*. Введём в него код программы, вычисляющей сумму аргументов, переданных в командной строке. Файл откроем редактором **Vim** (рис. 3.10).

```
%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 iprintfLF ; печать результата
call quit ; завершение программы
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --
```

6,8 All

Рис. 3.10: **Vim** с файлом *lab8-3.asm*

Теперь скомпилируем программу и проверим её работу (рис. 3.11).

```
[dianosov@arch lab08]$ nasm -f elf lab8-3.asm
[dianosov@arch lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[dianosov@arch lab08]$ ./lab8-3 1 2 4 8
Результат: 15
[dianosov@arch lab08]$ ./lab8-3 1 0 1 0
Результат: 2
[dianosov@arch lab08]$
```

Рис. 3.11: Компиляция и запуск *lab8-3.asm*

3.2.2 Вычисление произведения аргументов

Теперь изменим код так, чтобы программа выводила не сумму, а произведение аргументов командной строки. Для этого вместо

```
pop eax
call atoi
add esi, eax
```

напишем

```
pop eax          ; eax := следующий аргумент
call atoi
mov ebx, eax     ; ebx := eax
mov eax, esi     ; eax := esi
mul ebx          ; eax := eax * ebx
mov esi, eax     ; esi := eax = eax * ebx = esi * ebx
```

Также, изменим изначальное значение `esi` с 0 на 1 (нейтральный элемент по умножению, а не по сложению).

Сделаем изменения в файле программы и скомпилируем его. Проверим работу. (рис. 3.12, 3.13).


```

#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 ; eax := следующий аргумент
call atoi
mov ebx, eax ; ebx := eax
mov eax, esi ; eax := esi
mul ebx ; eax := eax * ebx
mov esi, eax ; esi := eax = eax * ebx = esi * ebx
; 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
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --

```

Рис. 3.12: **Vim** с измененным файлом *lab8-3.asm*

```

[dianosov@arch lab08]$ nasm -f elf lab8-3.asm
[dianosov@arch lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[dianosov@arch lab08]$ ./lab8-3 1 1 2
Результат: 2
[dianosov@arch lab08]$ ./lab8-3 1 1 2 213
Результат: 426
[dianosov@arch lab08]$ ./lab8-3 0 1 2 3
Результат: 0
[dianosov@arch lab08]$

```

Рис. 3.13: Компиляция и запуск *lab8-3.asm*

Как видно, программа правильно выводит произведение переданных в командной строке аргументов.

4 Задание для самостоятельной работы

Для начала, нужно выбрать вид функции $f(x)$ из таблицы. Варианту №6 соответствует функция

$$f(x) = 4x - 3$$

Создадим для выполнения задания файл *task.asm*, откроем его в редакторе **Vim** (рис. 4.1)

```

#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 ; eax := следующий аргумент
call atoi

; имеем аргумент x в eax
; требуется сделать из него 4x-3
mov ebx, 4
mul ebx ; eax := eax * 4 = 4x
sub eax, 3 ; eax := 4x - 3

add esi, eax ; esi := esi + 4x-3
mov eax, esi

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
~
~
~
~
~
"task.asm" 39L, 1443B
31,0-1 All

```

Рис. 4.1: Vim с файлом *task.asm*

Программа аналогична одной из уже рассмотренных в этой лабораторной работе. Теперь она суммирует (прибавляет к регистру *esi*) не сами аргументы (x_i), а значения функции, соответствующие им: $f(x_i)$.

Протестируем работу программы (рис. 4.2).

```

[dianosov@arch lab08]$ nasm -f elf task.asm
[dianosov@arch lab08]$ ld -m elf_i386 -o task task.o
[dianosov@arch lab08]$ ./task 1 2 3 4
Результат: 28
[dianosov@arch lab08]$

```

Рис. 4.2: Проверка работы *task.asm*

Выводится верный результат: $f(1) + f(2) + f(3) + f(4) =$

$$= 4 \times 1 - 3 + 4 \times 2 - 3 + 4 \times 3 - 3 + 4 \times 4 - 3 =$$

$$= 40 - 12 = 28$$

Теперь сделаем так, чтобы программа выводила в начале строку $f(x) = 4x - 3$. Откроем программу в **Vim** и проведём необходимые изменения (рис. 4.3). В секцию констант `.data` добавим нужное сообщение, а в самом начале секции `.text` выведем его, используя функцию `sprintf`.

```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg1 db "Функция: f(x)=4x-3",0
SECTION .text
global _start
_start:
mov eax, msg1
call sprintf

; 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 ; eax := следующий аргумент
call atoi

; имеем аргумент x в eax
; требуется сделать из него 4x-3
mov ebx, 4
mul ebx ; eax := eax * 4 = 4x
sub eax, 3 ; eax := 4x - 3

add esi, eax ; esi := esi + 4x-3
mov eax, esi

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
~
~
-- INSERT --
```

Рис. 4.3: **Vim** с обновленным файлом *task.asm*

Проверим же работу измененной программы (рис. 4.4).

```
[dianosov@arch lab08]$ nasm -f elf task.asm
[dianosov@arch lab08]$ ld -m elf_i386 -o task task.o
[dianosov@arch lab08]$ ./task 1 2 3 4
Функция: f(x)=4x-3
Результат: 28
[dianosov@arch lab08]$
```

Рис. 4.4: Тестирование программы *task.asm*

Программа работает корректно. Задание выполнено.

Загрузим файлы на GitHub (рис. 4.5).

```
[dianosov@arch arch-pc]$ git add .
[dianosov@arch arch-pc]$ git commit -am "add files for lab08"
On branch master
nothing to commit, working tree clean
[dianosov@arch arch-pc]$ git push origin master
Enumerating objects: 52, done.
Counting objects: 100% (52/52), done.
Delta compression using up to 12 threads
Compressing objects: 100% (45/45), done.
Writing objects: 100% (45/45), 1.23 MiB | 2.80 MiB/s, done.
Total 45 (delta 12), reused 1 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (12/12), completed with 6 local objects.
To https://github.com/exterminateddd/pc-course-2024-2025
734d02f..6c81bda master -> master
[dianosov@arch arch-pc]$
```

Рис. 4.5: Загрузка файлов на GitHub

5 Выводы

В ходе выполнения лабораторной работы были приобретены навыки написания программ с использованием циклов и обработкой аргументов командной строки.