

**Лабораторная работа №8.
Программирование цикла. Обработка
аргументов командной строки**

Простейший вариант

Диана Садова Алексеевна

Содержание

| | | |
|----------|---|-----------|
| 1 | Цель работы | 5 |
| 2 | Задание | 6 |
| 2.1 | Порядок выполнения лабораторной работы | 6 |
| 2.1.1 | Реализация циклов в NASM | 6 |
| 2.1.2 | Обработка аргументов командной строки | 11 |
| 3 | Теоретическое введение | 16 |
| 4 | Выполнение лабораторной работы | 17 |
| 4.1 | Напишите программу, которая находит сумму значений функции $f(x)$ для | 17 |
| 5 | Выводы | 19 |
| | Список литературы | 20 |

Список иллюстраций

| | | |
|------|--|----|
| 2.1 | Создаем каталог и проверяем его наличие | 6 |
| 2.2 | Вводим код | 7 |
| 2.3 | Создаем исполняемый файл | 8 |
| 2.4 | Изменяем текст программы | 8 |
| 2.5 | Вывод если число не четное | 9 |
| 2.6 | Вывод если число четное | 9 |
| 2.7 | Вносим изменения в текст программы | 10 |
| 2.8 | Создаем исполняемый файл и проверяем его работу | 10 |
| 2.9 | Вводим код | 12 |
| 2.10 | Создаем каталог и проверяем его наличие | 12 |
| 2.11 | Создаем исполняемый файл и запускаем его с аргументами . . . | 13 |
| 2.12 | Создаем каталог и проверяем его наличие | 13 |
| 2.13 | Вводим код | 14 |
| 2.14 | Вводим код | 14 |
| 2.15 | Изменяем текст программы | 15 |
| 2.16 | Создаем исполняемый файл и запускаем | 15 |
| 4.1 | Создаем программу | 18 |
| 4.2 | Создаем исполняемый файл и запускаем его | 18 |

Список таблиц

1 Цель работы

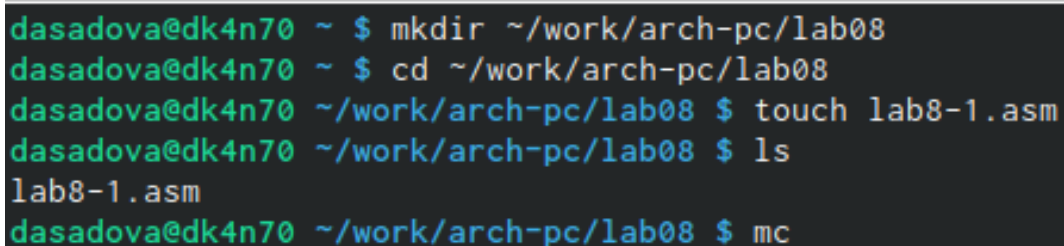
Приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки.

2 Задание

2.1 Порядок выполнения лабораторной работы

2.1.1 Реализация циклов в NASM

Создайте каталог для программ лабораторной работы № 8, перейдите в него и создайте файл lab8-1.asm:(рис. 2.1).



```
dasadova@dk4n70 ~ $ mkdir ~/work/arch-pc/lab08
dasadova@dk4n70 ~ $ cd ~/work/arch-pc/lab08
dasadova@dk4n70 ~/work/arch-pc/lab08 $ touch lab8-1.asm
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ls
lab8-1.asm
dasadova@dk4n70 ~/work/arch-pc/lab08 $ mc
```

Рис. 2.1: Создаем каталог и проверяем его наличие

Проверили, что файл создан правильно. Приступаем к следующему пункту лабораторной работы

При реализации циклов в NASM с использованием инструкции `loop` необходимо помнить о том, что эта инструкция использует регистр `ecx` в качестве счетчика и на каждом шаге уменьшает его значение на единицу. В качестве примера рассмотрим программу, которая выводит значение регистра `ecx`. Внимательно изучите текст программы (Листинг 8.1),(рис. 2.2)

Листинг 8.1. Программа вывода значений регистра `ecx`.

```

lab8-1.asm      [-M--]  9 L:[ 1+30 31/ 31] *(844 / 844b) <EOF>
;-----
; 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. 2.2: Вводим код

Введите в файл lab8-1.asm текст программы из листинга 8.1. Создайте исполняемый файл и проверьте его работу.(рис. 2.3).

```

dasadova@dk4n70 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 6
6
5
4
3
2
1
dasadova@dk4n70 ~/work/arch-pc/lab08 $

```

Рис. 2.3: Создаем исполняемый файл

Видим, что программы работает корректно

Данный пример показывает, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы. Измените текст программы добавив изменение значение регистра `ecx` в цикле:(рис. 2.4).

```

label:
sub ecx,1 ; 'ecx=ecx-1'
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения 'N'
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'

```

Рис. 2.4: Изменяем текст программы

Создайте исполняемый файл и проверьте его работу. Какие значения принимает регистр `ecx` в цикле? Соответствует ли число проходов цикла значению `N` введенному с клавиатуры?(рис. 2.5),(рис. 2.6).


```
4293727934
4293727932
4293727930
4293727928
4293727926
4293727924
4293727922
4293727920
4293727918
4293727916
4293727914
4293727912
4293727910
4293727908
4293727906
4293727904
4293727902
4293727900
4293727898
4293727896
4293727894
4293727892
4293727890
4293727888
4293727886
4293727884
42937278^C
dasadova@dk3n65 ~/work/arch-pc/lab08 $
```

Рис. 2.5: Вывод если число не четное

```
dasadova@dk4n70 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 6
5
3
1
dasadova@dk4n70 ~/work/arch-pc/lab08 $
```

Рис. 2.6: Вывод если число четное

Если мы вводим не четное значение, то программа выводит не корректное

значение. Если мы вводим четное число, программа выводит верные значения. В том и другом случае число N не соответствует числу проходов.

Для использования регистра `ecx` в цикле и сохранения корректности работы программы можно использовать стек. Внесите изменения в текст программы добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла(рис. 2.7).

```
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
```

Рис. 2.7: Вносим изменения в текст программы

Создайте исполняемый файл и проверьте его работу. Соответствует ли в данном случае число проходов цикла значению ☒ введенному с клавиатуры?(рис. 2.8).

```
dasadova@dk4n70 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 6
5
4
3
2
1
0
dasadova@dk4n70 ~/work/arch-pc/lab08 $
```

Рис. 2.8: Создаем исполняемый файл и проверяем его работу

Проверяем, что программа выполняется корректно

2.1.2 Обработка аргументов командной строки

При разработке программ иногда возникает необходимость указывать аргументы, которые будут использоваться в программе, непосредственно из командной строки при запуске программы.

При запуске программы в NASM аргументы командной строки загружаются в стек в обратном порядке, кроме того в стек записывается имя программы и общее количество аргументов. Последние два элемента стека для программы, скомпилированной NASM, – это всегда имя программы и количество переданных аргументов.

Таким образом, для того чтобы использовать аргументы в программе, их просто нужно извлечь из стека. Обработку аргументов нужно проводить в цикле. Т.е. сначала нужно извлечь из стека количество аргументов, а затем циклично для каждого аргумента выполнить логику программы. В качестве примера рассмотрим программу, которая выводит на экран аргументы командной строки. Внимательно изучите текст программы (Листинг 8.2).

Листинг 8.2. Программа выводящая на экран аргументы командной строки(рис. 2.9)

```

lab8-2.asm [-M--] 66 L:[ 1+ 2 3/ 23] *(207 /1151b) 0010 0x00A
;-----
; 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 sprintf ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку 'next')
_end:
call quit

```

Рис. 2.9: Вводим код

Создайте файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и введите в него текст программы из листинга 8.2.(рис. 2.10).

```

dasadova@dk4n70 ~/work/arch-pc/lab08 $ touch lab8-2.asm
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ls
in_out.asm lab8-1 lab8-1.asm lab8-1.o lab8-2.asm
dasadova@dk4n70 ~/work/arch-pc/lab08 $

```

Рис. 2.10: Создаем каталог и проверяем его наличие

Проверяем наличия файла в каталоге можем продолжать лабораторную работу
Создайте исполняемый файл и запустите его, указав аргументы:(рис. 2.11).

```

dasadova@dk4n70 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
dasadova@dk4n70 ~/work/arch-pc/lab08 $

```

Рис. 2.11: Создаем исполняемый файл и запускаем его с аргументами

Сколько аргументов было обработано программой?

Программой было выведено 4 аргумента: 1 строка - значение “аргумента1” был вписано без пробела, вывилось на экран как один аргумент, строка 2 и 3 - значение “аргумент 2” было вписано с пробелом, вывелось как два аргумента, 4 строка - значение “ ‘аргумент3’ ” изначально вводилось в скобках и программой распознавался как один аргумент.

Рассмотрим еще один пример программы которая выводит сумму чисел, которые пере даются в программу как аргументы. Создайте файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 и введите в него текст программы из листинга 8.3.(рис. 2.12).

```

dasadova@dk4n70 ~/work/arch-pc/lab08 $ touch lab8-3.asm
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ls
in_out.asm lab8-1 lab8-1.asm lab8-1.o lab8-2 lab8-2.asm lab8-2.o lab8-3.asm
dasadova@dk4n70 ~/work/arch-pc/lab08 $

```

Рис. 2.12: Создаем каталог и проверяем его наличие

После проверки наличия файла в каталоге можем продолжать лабораторную работу

Листинг 8.3. Программа вычисления суммы аргументов командной строки(рис. 2.13).

```

lab8-3.asm      [-M--] 32 L:[ 1+28 29/ 29] *(1428/1428b) <EOF>
#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. 2.13: Вводим код

Создайте исполняемый файл и запустите его, указав аргументы. Пример результата работы программы:(рис. 2.14).

```

dasadova@dk4n70 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47
dasadova@dk4n70 ~/work/arch-pc/lab08 $

```

Рис. 2.14: Вводим код

Измените текст программы из листинга 8.3 для вычисления произведения

аргументов командной строки.(рис. 2.15).(рис. 2.16).

```
lab8-3.asm [----] 11 L:[ 1+22 23/ 31] *(1004/1395b) 0010 0x00A
#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 ; умножаем
mov esi,eax
; 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. 2.15: Изменяем текст программы

```
dasadova@dk3n65 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
dasadova@dk3n65 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
dasadova@dk3n65 ~/work/arch-pc/lab08 $ ./lab8-3 3 4 8
Результат: 96
dasadova@dk3n65 ~/work/arch-pc/lab08 $
```

Рис. 2.16: Создаем исполняемый файл и запускаем

Видим, что программа выполняется корректно

3 Теоретическое введение

4 Выполнение лабораторной работы

4.1 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 № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах $x = x_1, x_2, \dots, x_n$. Пример работы программы для функции $f(x) = x + 2$ и набора $x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4$:

```
user@dk4n31:~$ ./main 1 2 3 4 Функция: f(x)=x+2 Результат: 18 user@dk4n31:~$
```

(рис. 4.1).(рис. 4.2).

Мой вариант 19. При решении этого задания используем полученные навыки при решении лабораторной работы и данные 19 варианта.

```

lab8-zadanie.asm  [----] 21 L: [ 1+ 0 1/ 35] *(21 /1522b) 0010 0x00A
#include 'in_out.asm'
SECTION .data
msg1 db "Функция: f(x)=8*x-3",0
msg db "Результат: ",0
SECTION .text
global _start
_start:
mov eax,msg1
call sprintf
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
mov ebx,8
mul ebx
sub eax,3 ; добавляем к промежуточной сумме
add esi,eax
; след. аргумент 'esi=esi+eax'
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprintf
mov eax, esi ; записываем сумму в регистр 'eax'
call iprintf ; печать результата
call quit ; завершение программы

```

Рис. 4.1: Создаем программу

```

dasadova@dk4n70 ~/work/arch-pc/lab08 $ nasm -f elf lab8-zadanie.asm
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-zadanie lab8-zadanie.o
dasadova@dk4n70 ~/work/arch-pc/lab08 $ ./lab8-zadanie 1 2 3 4
Функция: f(x)=8*x-3
Результат: 68
dasadova@dk4n70 ~/work/arch-pc/lab08 $

```

Рис. 4.2: Создаем исполняемый файл и запускаем его

5 Выводы

Мы приобрели навыки написания программ с использованием циклов и обработки аргументов командной строки.

Список литературы