

Лабораторная работа №9

Программирование цикла. Обработка аргументов командной строки.

Татьяна Алексеевна Коннова, НПИбд-01-22

Содержание

1	Цель работы	4
1.1	Задание	4
1.2	Выполнение лабораторной работы	4
1.2.1	Реализация циклов в NASM	4
1.3	9.3.2. Обработка аргументов командной строки	11
1.4	Самостоятельная работа	15
2	Выводы	18

Список иллюстраций

1.1	lab9_1	5
1.2	lab9_1	6
1.3	lab9_1	7
1.4	lab9_1	8
1.5	lab9_1	9
1.6	lab9_1	10
1.7	lab9_1	11
1.8	lab9_2	12
1.9	lab9_2	12
1.10	lab9_3	13
1.11	lab9_3	13
1.12	lab9_3	14
1.13	lab9_3	14
1.14	samost_rab_lab9_4	16
1.15	samost_rab_lab9_4	17

1 Цель работы

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

1.1 Задание

Программирование ветвлений

1.2 Выполнение лабораторной работы

1.2.1 Реализация циклов в NASM

Создаем файл lab9-1.asm touch lab9-1.asm 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 (Листинг 9.1). Введем в файл lab9-1.asm текст программы из листинга 9.1. Создадим исполняемый файл и проверим его работу. Данный пример показывает, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы. (рис. 1.1)

(рис. 1.2)

```
mc [takonnova@fedora]:~/work/study/2022-2
lab9-1.asm [-M--] 9 L:[ 1+30 31/ 31]
;-----
; 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. 1.1: lab9_1

```
[takonnova@fedora lab09]$ nasm -f elf lab9-1.asm
[takonnova@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[takonnova@fedora lab09]$ ./lab9-1
Введите N: 8
8
7
6
5
4
3
2
1
```

Рис. 1.2: lab9_1

Изменим текст программы, добавив изменение значение регистра `ecx` в цикле:
`label:...`

(рис. 1.3)

```
mc [takonnova@fedora]:~/work/study/2022-2023
lab9-1.asm [-M--] 9 L:[ 1+31 32/ 32]
;-----
; 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:
sub ecx,1
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. 1.3: lab9_1

Создаем исполняемый файл и проверяем его работу.

(рис. 1.4)

(рис. 1.5)

```
4294938312
4294938310
4294938308
4294938306
4294938304
4294938302
4294938300
4294938298
4294938296
4294938294
4294938292
4294938290
4294938288
4294938286
4294938284
4294938282
4294938280
4294938278
4294938276
4294938274
4294938272
4294938270
4294938268
4294938266
4294938264
4294938262
4294938260
4294938258
42949^C
[takonnova@fedora lab09]$
```

Рис. 1.4: lab9_1

При четных:(создала еще один новый файл lab9_1.asm, так как забыла про отделение четных чисел и нечетных)


```
[takonnova@fedora lab09]$ ./lab9_1
Введите N: 6
5
3
1
[takonnova@fedora lab09]$
```

Рис. 1.5: lab9_1

Соответствует ли число проходов цикла значению N введенному с клавиатуры?

- Ответ: бесконечный цикл выводится при нечетных N и нечетные числа выводятся при четных значениях N

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

Создаем исполняемый файл и проверяем его работу. (рис. 1.6) (рис. 1.7) Соответствует ли в данном случае число проходов цикла значению N введенному с клавиатуры?

- Ответ: программа обрабатывает, выводит числа от N-1 до 0, поэтому число проходов цикла есть само число N раз.

```

lab9-1.asm      [-M--]   8 L:[  1+31  32/ 34
;-----
; 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:
push ecx ;
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
; переход на `label`
pop ecx.
loop label
call quit

```

Рис. 1.6: lab9_1

```

[takonnova@fedora lab09]$ nasm -f elf lab9-1.asm
[takonnova@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[takonnova@fedora lab09]$ ./lab9-1
Введите N: 9
8
7
6
5
4
3
2
1
0
[takonnova@fedora lab09]$

```

Рис. 1.7: lab9_1

1.3 9.3.2. Обработка аргументов командной строки

При разработке программ иногда встает необходимость указывать аргументы, которые будут использоваться в программе, непосредственно из командной строки при запуске программы. 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 изучим текст программы (Листинг 9.2).

Создаем файл lab9-2.asm в каталоге ~/work/arch-pc/lab09 и введите в него текст программы из листинга 9.2. Создаем исполняемый файл и запускаем его, указав аргументы (рис. 1.8) (рис. 1.9)

```
mc [takonnova@fedora]:~/work/study/2022-2023/Ar
lab9-2.asm [----] 66 L: [ 1+ 2 3/ 23] *(20
;-----
; 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. 1.8: lab9_2

```
[takonnova@fedora lab09]$ nasm -f elf lab9-2.asm
[takonnova@fedora lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[takonnova@fedora lab09]$ ./lab9-2 Аргумент#1 1 Аргумент#2 2 'Аргумент#3'
Аргумент#1
1
Аргумент#2
2
Аргумент#3
[takonnova@fedora lab09]$
```

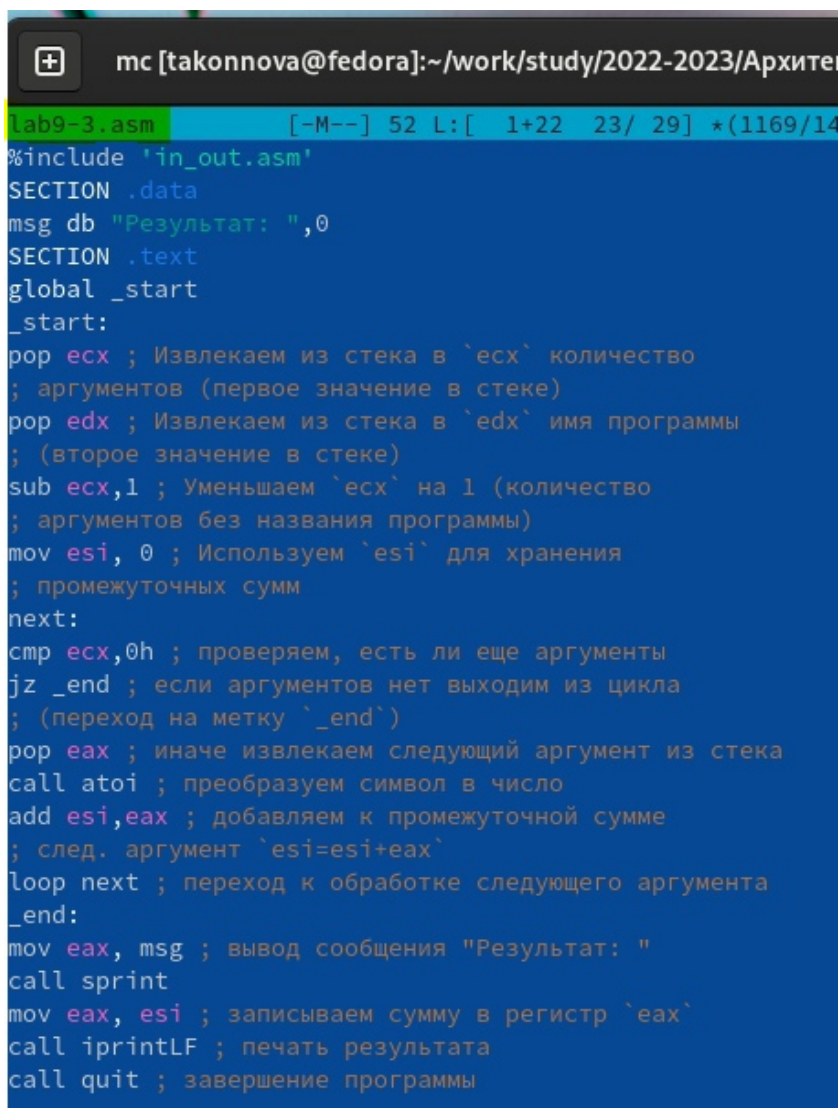
Рис. 1.9: lab9_2

Сколько аргументов было обработано программой?

* Ответ: 5

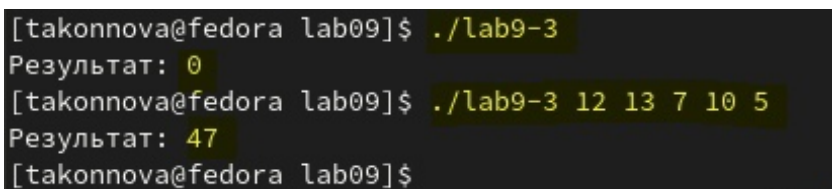
Рассмотрим еще один пример программы которая выводит сумму чисел, кото-
рые передаются в программу как аргументы. Создайте файл lab9-3.asm, введите

в него текст программы из листинга 9.3. Создайте исполняемый файл и запустите его, указав аргументы. (рис. 1.10) (рис. 1.11)



```
mc [takonnova@fedora]:~/work/study/2022-2023/Архитек
lab9-3.asm [-M--] 52 L:[ 1+22 23/ 29] *(1169/14
%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. 1.10: lab9_3



```
[takonnova@fedora lab09]$ ./lab9-3
Результат: 0
[takonnova@fedora lab09]$ ./lab9-3 12 13 7 10 5
Результат: 47
[takonnova@fedora lab09]$
```

Рис. 1.11: lab9_3

Изменим текст программы из листинга 9.3 для вычисления произведения аргументов командной строки.

(рис. 1.12) (рис. 1.13)

```
lab9-3.asm [----] 32 L: [ 1+31 32/ 32] *(1460/14
#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
mov ebx,eax
mov eax,esi
mul ebx
mov 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. 1.12: lab9_3

```
[takonnova@fedora lab09]$ ./lab9-3 4 7 1 8
Результат: 224
[takonnova@fedora lab09]$
```

Рис. 1.13: lab9_3

1.4 Самостоятельная работа

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 9.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы No 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах $x = x_1, x_2, \dots, x_n$.

Вариант моей самостоятельной работы = №15, а именно $f(x) = 6x + 13$

(рис. 1.14) (рис. 1.15)


```

mc [takonnova@fedora]:~/work/stu
lab9-4.asm [----] 0 L: [ 1+38
%include 'in_out.asm'

SECTION .data
msg1 db "Функция: f(x) = 6x + 13", 0
msg db "Результат: ", 0

SECTION .text
global _start

_start:

mov eax, msg1
call sprintLF

pop ecx
pop edx

sub ecx, 1
mov esi, 0
next:
cmp ecx, 0h
jz _end.
pop eax
call atoi

mov ebx, 6
mul ebx
add eax, 13

add esi, eax
loop next

_end:
mov eax, msg.
call sprint
mov eax, esi.
call iprintLF.
call quit.

```

Рис. 1.14: samost_rab_lab9_4


```
[takonnova@fedora lab09]$ ./lab9-4 1 2 3 4
Функция:  $f(x) = 6x + 13$ 
Результат: 112
```

Рис. 1.15: samost_rab_lab9_4

2 Выводы

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