

Отчет по лабораторной работе №8

Шубина София Антоновна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Задание для самостоятельной работы	15
4	Выводы	18
	Список литературы	19

Список иллюстраций

2.1	Создание файла и каталога	6
2.2	Написание текста программы	7
2.3	Создание исполняемого файла и проверка его работы	7
2.4	Изменение текста программы	8
2.5	Создание исполняемого файла и проверка его работы	9
2.6	Изменение текста программы	10
2.7	Создание исполняемого файла и проверка его работы	10
2.8	Создание файла	11
2.9	Написание текста программы	12
2.10	Создание и проверка исполняемого файла	12
2.11	Создание файла	13
2.12	Написание текста программы	13
2.13	Изменение текста программы	14
2.14	Создание и проверка исполняемого файла	14
3.1	Написание текста программы	16
3.2	Создание и проверка исполняемого файла	16

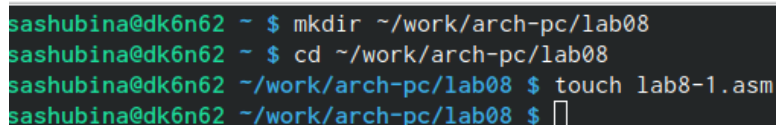
Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

Реализация циклов в NASM Создадим каталог для программ лабораторной работы № 8, перейдем в него и создадим файл lab8-1.asm: `mkdir ~/work/arch-pc/lab08 cd ~/work/arch-pc/lab08 touch lab8-1.asm` (рис. 2.1).



```
sashubina@dk6n62 ~ $ mkdir ~/work/arch-pc/lab08
sashubina@dk6n62 ~ $ cd ~/work/arch-pc/lab08
sashubina@dk6n62 ~/work/arch-pc/lab08 $ touch lab8-1.asm
sashubina@dk6n62 ~/work/arch-pc/lab08 $
```

Рис. 2.1: Создание файла и каталога

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 изучим текст программы Введем в файл `lab8-1.asm` текст программы из листинга. Создадим исполняемый файл и проверим его работу.(рис. 2.2) (рис. 2.3).

```

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: Написание текста программы

```

sashubina@dk6n62 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
sashubina@dk6n62 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.asm
ld:lab8-1.asm: file format not recognized; treating as linker script
ld:lab8-1.asm:1: syntax error
sashubina@dk6n62 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
sashubina@dk6n62 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
5
4
3
2
1
sashubina@dk6n62 ~/work/arch-pc/lab08 $

```

Рис. 2.3: Создание исполняемого файла и проверка его работы

Данный пример показывает, что использование регистра ecx в теле цикла loop может привести к некорректной работе программы. Изменим текст программы добавив изменение значение регистра ecx в цикле: Создадим исполняемый файл и проверим его работу. Какие значения принимает регистр ecx в цикле?

Использование регистра `eax` в теле цикла `loop` приводит к некорректной работе программы. Соответствует ли число проходов цикла значению ☒ введенному с клавиатуры? Нет, число проходов намного больше заявленного `N`/. (рис. 2.4) (рис. 2.5).

```
lab8-1.asm      [-M--] 24 L: [ 1+25 26/ 33] *(705 / 870b) 0010 0x00A
;~~~~~
; Программа вывода значений регистра 'ecx'
;~~~~~
#include "io.h"
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 ; 'ecx=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. 2.4: Изменение текста программы


```
4293303352
4293303350
4293303348
4293303346
4293303344
4293303342
4293303340
4293303338
4293303336
4293303334
4293303332
4293303330
4293303328
4293303326
4293303324
4293303322
4293303320
4293303318
4293303316
4293303314
4293303312
4293303310
4293303308
4293303306
4293303304
4293303302
4293303300
4293303298
4293303296
4293303294
4293303292
4293303290
4293303288
4293303286
4293303284
4293303282
4293303280
4293303278
4293303276
4293303274
4293303272
4293303270^Z
[[!]+ Остановлен ./lab8-1
sashubina@dk6n62 ~/work/arch-pc/lab08 $ █
```

Рис. 2.5: Создание исполняемого файла и проверка его работы

Для использования регистра `ecx` 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`: Создадим исполняемый файл и проверим его работу. Соответствует ли в данном случае число проходов цикла значению ☒ введенному с клавиатуры? В данном случае число проходов соответствует введенному числу N . (рис. 2.6) (рис. 2.7).

```

lab8-1.asm      [---] 11 L: [ 1+31 32/ 34] *(888 / 899b) 0010 0x00A
; Програма вивода значень реєстра 'ecx'
; Програма вивода значень реєстра 'ecx'
%include "in_out.asm"
SECTION .data
msg1 db "Введіть N: ",0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:
; ----- Вивод повідомлення 'Введіть 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 ; добавление значений ecx в стек
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintfLF
pop ecx ; извлечение значения ecx из стека
loop label
call quit

```

Рис. 2.6: Изменение текста программы

```

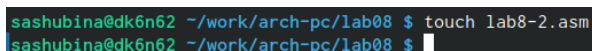
sashubina@dk6n62 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
sashubina@dk6n62 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
sashubina@dk6n62 ~/work/arch-pc/lab08 $ ./lab8-1
Введіть N: 5
4
3
2
1
0
sashubina@dk6n62 ~/work/arch-pc/lab08 $

```

Рис. 2.7: Создание исполняемого файла и проверка его работы

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 обрат- ном порядке, кроме того в стек записывается имя программы и общее количество аргументов. Последние два элемента стека для программы, ском- пилированной NASM, – это всегда имя программы и количество переданных аргументов. Таким образом, для того чтобы использовать аргументы в програм- ме, их просто нужно извлечь из стека. Обработку аргументов нужно проводить в цикле. Т.е. сначала нужно из- влечь из стека количество аргументов, а затем цик- лически для каждого аргумента выполнить логику программы. В качестве примера рассмотрим программу, которая выводит на экран аргументы командной строки. Внимательно изучим текст программы . Программа выводящая на экран аргумен- ты командной строки Создадим файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и введем в него текст про- граммы из листинга. Создадим исполняемый файл и запустим его, указав аргументы: user@dk4n31:~\$./lab8-2 аргумент1 аргумент 2 ‘аргумент 3’ Сколько аргументов было обработано программой? 4 аргумента обработано программой (рис. 2.8) (рис. 2.10) (рис. ??).



```
sashubina@dk6n62 ~/work/arch-pc/lab08 $ touch lab8-2.asm
sashubina@dk6n62 ~/work/arch-pc/lab08 $
```

Рис. 2.8: Создание файла

```
lab8-2.asm [----] 9 L: [ 1+22 23/ 23] *(1151/1151b) <EOF>
;~~~~~
; Обработка аргументов командной строки
;~~~~~
%include "lab00.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 аргумента
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: Написание текста программы

```
sashubina@dk6n62 ~/work/arch-pc/lab08 $ ./lab8-2
sashubina@dk6n62 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
sashubina@dk6n62 ~/work/arch-pc/lab08 $
```

Рис. 2.10: Создание и проверка исполняемого файла

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. Создадим файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 и введем в него текст программы из листинга. Создадим исполняемый файл и запустим его, указав аргументы. Пример результата работы программы: user@dk4n31:~\$./main 12 13 7 10 5 Результат: 47 user@dk4n31:~\$ (рис. 2.11) (рис. 2.12) (рис. ??).

```
sashubina@dk6n62 ~/work/arch-pc/lab08 $ touch lab8-3.asm
sashubina@dk6n62 ~/work/arch-pc/lab08 $
```

Рис. 2.11: Создание файла

```
lab8-3.asm      [-M--] 32 L:[ 1+28 29/ 29] *(1428/1428b) <EOF>
%include "lab8-3.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,ecx ; добавляем к промежуточной сумме
; 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 iprintLF ; печать результата
call quit ; завершение программы
```

Рис. 2.12: Написание текста программы

```
sashubina@dk6n62 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
sashubina@dk6n62 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
sashubina@dk6n62 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47
sashubina@dk6n62 ~/work/arch-pc/lab08 $
```

Изменим текст про-

граммы из листинга для вычисления произведения аргументов командной строки. (рис. 2.13) (рис. 2.14).

```

lab8-3.asm [----] 11 L: [ 1+21 22/ 32] *(999 /1392b) 0010 0x00A
%include "lab8-3.inc"
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 произведений
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,ecx
; 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 iprintf ; печать результата
call quit ; завершение программы

```

Рис. 2.13: Изменение текста программы

```

sashubina@dk6n62 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
sashubina@dk6n62 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
sashubina@dk6n62 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 54600
sashubina@dk6n62 ~/work/arch-pc/lab08 $

```

Рис. 2.14: Создание и проверка исполняемого файла

3 Задание для самостоятельной работы

1. Напишем 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 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы No 7. ВАРИАНТ 6 Создадим исполняемый файл и проверим 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. 3.1) (рис. 3.2).

```
Func.asm [----] 3 L: [ 1+18 19/ 31] *(221 / 327b) 0032 0x020
%include "in_out.asm"
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi,0
next:
cmp ecx,0h
jz _end
pop eax
call atoi
mov ebx,4
mul ebx
mov ebx,3
sub eax,ebx
add esi,eax
loop next
_end:
mov eax,msg
call sprint
mov eax,esi
call iprintLF
call quit
```

Рис. 3.1: Написание текста программы

```
sashubina@dk6n62 ~/work/arch-pc/lab08 $ nasm -f elf func.asm
sashubina@dk6n62 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o func func.o
sashubina@dk6n62 ~/work/arch-pc/lab08 $ ./func 1 2 3 4
Результат: 28
sashubina@dk6n62 ~/work/arch-pc/lab08 $
```

Рис. 3.2: Создание и проверка исполняемого файла

```
%include 'in_out.asm'

SECTION .data
msg db "Результат: ",0

SECTION .text
global _start
_start:
pop ecx
```



```
pop edx
sub ecx,1
mov esi,0
next:
cmp ecx,0h
jz _end
pop eax
call atoi
mov ebx,4
mul ebx
mov ebx,3
sub eax,ebx

add esi,eax

loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```

4 Выводы

Я приобрела теоретические и практические навыки написания программ с использованием циклов и обработкой аргументов командной строки.

Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learning-bash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс,
- 11.
12. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
13. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
14. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
15. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-

- е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
16. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
17. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер,
18. — 1120 с. — (Классика Computer Science).