

Отчёт по лабораторной работе №7

По теме: Команды безусловного и условного переходов в NASM. Программирование ветвлений.

Выполнил: Чубаев Кирилл Евгеньевич, НММбд-04-24

Содержание

| | |
|--|----|
| Цель работы | 1 |
| Выполнение лабораторной работы | 1 |
| Выполнение самостоятельной работы..... | 8 |
| Вывод | 14 |
| Список литературы | 14 |

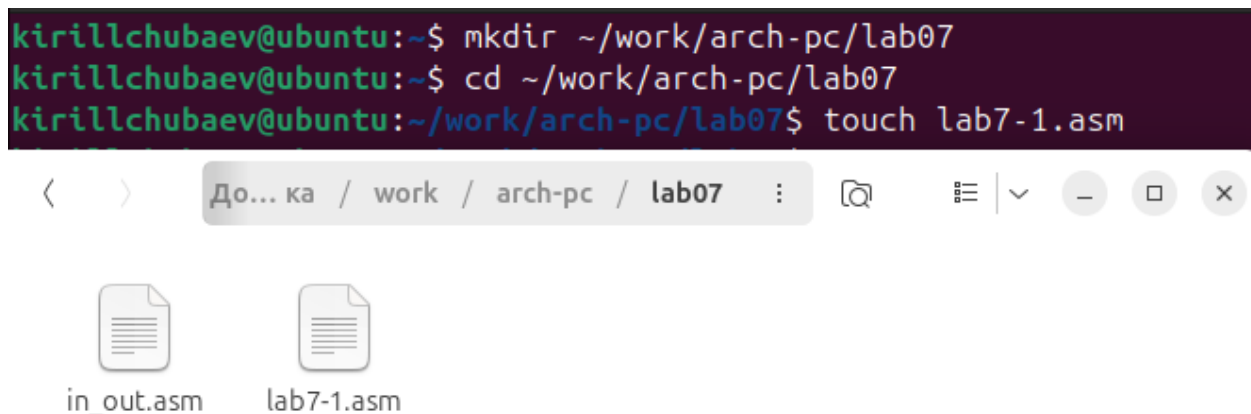
Цель работы

Целью данной лабораторной работы является изучение команд условного и безусловного переходов, приобретение навыков написания программ с использованием переходов, а также знакомство с назначением и структурой файла листинга.

Выполнение лабораторной работы

1. Я создал каталог lab07 в директории "~/work/arch-pc". Далее создал файл lab7-1.asm внутри каталога:

```
kirillchubaev@ubuntu:~$ mkdir ~/work/arch-pc/lab07
kirillchubaev@ubuntu:~$ cd ~/work/arch-pc/lab07
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ touch lab7-1.asm
```



2. Далее ввел в файл текст программы из листинга 7.1:

```

%include 'in_out.asm'|
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label2

_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'

_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'

_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'

_end:
call quit ; вызов подпрограммы завершения

```

3. Я создал исполняемый файл и запустил его. Результат соответствовал нужному:

```

kirillchubaev@ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ █

```

4. Я изменил текст программы с помощью листинга 7.2 таким образом, чтобы она сначала выводила 'Сообщение № 2', потом 'Сообщение № 1' и завершала работу. С помощью листинга 7.2 я написал код. Далее проверил его работу. На экран вывелся следующий результат:

```
%include 'in_out.asm'
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label2

_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
jmp _end

_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
jmp _label1

_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'|

_end:
call quit ; вызов подпрограммы завершения
```

```

kirillchubaev@ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ █

```

5. Я изменил текст программы так, чтобы сначала выводило 'Сообщение №3', затем 'Сообщение №2', затем 'Сообщение №1':

```

%include 'in_out.asm'
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label3

_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
jmp _end

_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
jmp _label1

_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
jmp _label2

_end:
call quit ; вызов подпрограммы завершения

```

6. Далее запустил программу и проверил ее работу. Программа работает корректно. На экран вывелось следующее:

```
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
kirillchubaev@ubuntu:~/work/arch-pc/lab07$
```

7. Я создал файл lab7-2.asm. Далее внимательно изучил текст программы из листинга 7.3 и написал текст программы:

```
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ touch lab7-2.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab07$
```



```
%include 'in_out.asm'
.....
section .data
msg1 db 'Введите B: ',0h
msg2 db "Наибольшее число: ",0h
A dd '20'
C dd '50'
section .bss
max resb 10
B resb 10
section .text
global _start
_start:

mov eax,msg1
call sprint

mov ecx,B
mov edx,10
call sread

mov eax,B
call atoi
mov [B],eax

mov ecx,[A]
mov [max],ecx

cmp ecx,[C]
jg check_B
mov ecx,[C]
```

```

mov [max],ecx

check_B:
mov eax,max
call atoi
mov [max],eax

mov ecx,[max]
cmp ecx,[B]
jg fin
mov ecx,[B]
mov [max],ecx

fin:
mov eax, msg2
call sprint
mov eax,[max]
call iprintLF
call quit

```

8. Я создал исполняемый файл и запустил программу. Далее ввел несколько разных чисел, чтобы проверить, как работает программа. Программа работает корректно:

```

kirillchubaev@ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 7
Наибольшее число: 50
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 65
Наибольшее число: 65
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 89
Наибольшее число: 89
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 34
Наибольшее число: 50

```

9. Я создал файл листинга lab7-2.lst и открыл его.

```

kirillchubaev@ubuntu:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ mcedit lab7-2.lst

```

10. Проанализировав файл, я понял как он работает и какие значения выводит:
- 1) Эта строка находится на 21 месте, ее адрес "00000101", Машинный код - B8[0A000000], а "mov eax, B" - исходный текст программы, означающий, что в регистр eax мы вносим значения переменной B.

```
21 00000101 B8[0A000000]          mov eax,B
```

- 2) Эта строка находится на 35 месте, ее адрес - "00000135", Машинный код - E862FFFFFF, а функция "call atoi" - исходный текст программы, означающий, что символ, лежащий в строке выше, переводится в число.

```
35 00000135 E862FFFFFF          call atoi.
```

- 3) Эта строка находится на 47 месте, ее адрес - "00000163", Машинный код - A1[00000000], а "mov eax, [max]" - исходный текст программы, означающий, что число, хранившееся в переменной max, записывается в регистр eax.

```
47 00000163 A1[00000000]          mov eax,[max]
```

11. В строке "mov eax, max" я убрал "max" и попробовал создать файл. В терминале вывелась ошибка, так как для программа требует два операнда:

```
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:34: error: invalid combination of opcode and operands
```

12. В файле листинга показывается, где именно ошибка и с чем она связана:

```
34                                mov eax
34  *****                      error: invalid combination of opcode and operands
```

Выполнение самостоятельной работы

1. Для выполнения первого задания самостоятельной работы я сначала создал файл lab7-test1.asm:

```
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ touch lab7-test1.asm
```

2. Далее я написал программу для нахождения наименьшего числа из трех введенных в терминал:


```
%include 'in_out.asm'
```

```
SECTION .data
```

```
A1 DB 'Введите число A: ',0h
```

```
B1 DB 'Введите число B: ',0h
```

```
C1 DB 'Введите число C: ',0h
```

```
res DB 'Наименьшее число: ',0h
```

```
SECTION .bss
```

```
min RESB 20
```

```
A RESB 20
```

```
B RESB 20
```

```
C RESB 20
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov eax,A1
```

```
call sprint
```

```
mov ecx,A
```

```
mov edx,20
```

```
call sread
```

```
mov eax, A
```

```
call atoi
```

```
mov [A],eax
```

```
xor eax,eax
```

```
mov eax,B1  
call sprint
```

```
mov ecx,B  
mov edx,20  
call sread
```

```
mov eax,B  
call atoi  
mov [B],eax
```

```
xor eax,eax
```

```
mov ecx, [A]  
mov [min],ecx  
mov ecx,[min]
```

```
cmp ecx,[B]  
jl check_C  
mov ecx, [B]  
mov [min],ecx
```

```
check_C:
```

```
mov eax,C1  
call sprint
```

```
mov ecx,C  
mov edx,10  
call sread
```

```

mov eax,C
call atoi
mov [C],eax

xor eax,eax

mov ecx,[min]
cmp ecx,[C]
jl fin
mov ecx,[C]
mov [min],ecx

fin:
mov eax, res
call sprint
mov eax, [min]
call iprintLF
call quit

```

3. В лабораторной работе №6 я получил вариант 14, поэтому я должен ввести следующие числа: 81, 22, 72. Я создал исполняемый файл и запустил программу. Программа работает корректно и вывелся следующий результат:

```

kirillchubaev@ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-test1.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-test1 lab7-test1.o
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ./lab7-test1
Введите число A: 81
Введите число B: 22
Введите число C: 72
Наименьшее число: 22
kirillchubaev@ubuntu:~/work/arch-pc/lab07$

```

4. Для выполнения второго задания самостоятельной работы я создал файл lab7-test2.asm:

```

kirillchubaev@ubuntu:~/work/arch-pc/lab07$ touch lab7-test2.asm

```

5. Далее я написал программу, чтобы она вычисляла необходимое выражение при введенных "a" и "x". Поскольку мой вариант - 14, то я написал код для вычисления выражения из 14 варианта:

```

%include 'in_out.asm'

SECTION .data
prim1 DB '3*a+1, x<a' ,0
prim2 DB '3*x+1, x=>a' ,0
X1 DB 'Введите значение x:',0
A1 DB 'Введите значение a:',0
otv DB 'Ответ: ',0

SECTION .bss
X RESB 20
A RESB 20
F RESB 20
SECTION .text
GLOBAL _start
_start:

mov eax,prim1
call sprintf
mov eax,prim2
call sprintf

mov eax,X1
call sprintf

mov ecx,X
mov edx,10
call sread

mov eax,X

```

```
call atoi
mov [X],eax

mov eax,A1
call sprint

mov ecx,A
mov edx,10
call sread

mov eax,A
call atoi
mov [A],eax

mov ecx,[X]
mov [F],ecx

cmp ecx,[A]
jl less_than_a
mov eax, [X]
mov ebx, 3
mul ebx
add eax, 1
mov [F], eax
jmp fin

less_than_a:
mov eax, [A]
mov ebx, 3
mul ebx
```

```
add eax, 1
mov [F], eax

fin:
mov eax,otv
call sprint
mov eax,[F]
call iprintLF
call quit
```

6. Я создал исполняемый файл и запустил программу. Далее ввёл необходимые значения “a” и “x” из варианта 14. Программа работает корректно и выводит правильные вычисления:

```
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-test2.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-test2 lab7-test2.o
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ./lab7-test2
3*a+1, x<a
3*x+1, x=>a
Введите значение x:2
Введите значение a:3
Ответ: 10
kirillchubaev@ubuntu:~/work/arch-pc/lab07$ ./lab7-test2
3*a+1, x<a
3*x+1, x=>a
Введите значение x:4
Введите значение a:2
Ответ: 13
kirillchubaev@ubuntu:~/work/arch-pc/lab07$
```

Вывод

В ходе данной лабораторной работы я изучил команды условного и безусловного перехода, а также приобрел полезные навыки написания программ с переходами.

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/Learningbash-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, 2017.
11. *Новожилов О. П.* Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
13. *Робачевский А., Немнюгин С., Стесик О.* Операционная система UNIX. — 2-е изд. — БХВ-Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. *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.
15. *Таненбаум Э.* Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. *Таненбаум Э., Бос Х.* Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).