

Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютеров и операционные системы

Лисенков Егор Романович

Содержание

1	Цель работы	1
2	Задание.....	1
3	Теоретическое введение	1
4	Выполнение лабораторной работы	2
4.1	Реализация переходов в NASM	2
4.2	Изучение структуры файлы листинга	5
4.3	Задания для самостоятельной работы	7
5	Выводы.....	8
6	Список литературы.....	8

1 Цель работы

Изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

2 Задание

1. Реализация переходов в NASM.
2. Изучение структуры файлы листинга.
3. Задания для самостоятельной работы.

3 Теоретическое введение

Для реализации ветвлений в ассемблере используются так называемые команды передачи управления или команды перехода. Можно выделить 2 типа переходов:

- условный переход – выполнение или не выполнение перехода в определенную точку программы в зависимости от проверки условия.

- безусловный переход – выполнение передачи управления в определенную точку программы без каких-либо условий.

Безусловный переход выполняется инструкцией `jmp`. Инструкция `str` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция `str` является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания.

Листинг (в рамках понятийного аппарата NASM) — это один из выходных файлов, создаваемых транслятором. Он имеет текстовый вид и нужен при отладке программы, так как кроме строк самой программы он содержит дополнительную информацию.

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

Создаю каталог для программ № 7, перехожу в него и создаю файл `lab7-1.asm`. (рис. ??).

```
erlisenkov@dk3n33 ~ $ mkdir ~/work/arch-pc/lab07
erlisenkov@dk3n33 ~ $ cd ~/work/arch-pc/lab07
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ touch lab7-1.asm
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $
```

Создание файлов для лабораторной работы

Ввожу в файл `lab7-1.asm` текст программы из листинга 7.1. (рис. ??).

```

lab7-1.asm      [-M--] 41 L:[ 1+19 20/ 20] *(6
%include 'in_out.asm' ; подключение внешнего файла
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 sprintfLF ; 'Сообщение № 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintfLF ; 'Сообщение № 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintfLF ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения

```

Ввод текста программы из листинга 7.1

Создаю исполняемый файл и запускаю его. (рис. ??).

```

erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ ld -m elf_i386 lab7-1.o -o lab7-1
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 3
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $

```

Запуск программного кода

Использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения.

Изменяю программу таким образом, чтобы она выводила сначала 'Сообщение № 2', потом 'Сообщение № 1' и завершала работу. Для этого ввожу команду из листинга. (рис. ??).

```

>> lab7-1.asm [-M--] 41 L: [ 1+21 22/ 22]
#include 'in_out.asm' ; подключение внешнего фай
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

```

Изменение текста программы

Создаю исполняемый файл и проверяю его работу. (рис. ??).

```

erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ ld -m elf_i386 lab7-1.o -o lab7-1
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 1
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $

```

Создание исполняемого файла

Затем изменяю текст программы, добавив в начале программы `jmp _label3`, `jmp _label2` в конце метки `jmp _label3`, `jmp _label1` добавляю в конце метки `jmp _label2`, и добавляю `jmp _end` в конце метки `jmp _label1`, (рис. ??).

Изменение текста программы

Изменение текста программы

чтобы вывод программы был следующим: (рис. ??).

```
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ ld -m elf_i386 lab7-1.o -o lab7-1
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

Вывод программы

Рассмотрим программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А,В и С. Значения для А и С задаются в программе, значение В вводится с клавиатуры.

Создаю файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. (рис. ??).

```
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ touch lab7-2.asm
```

Создание файла

Текст программы из листинга 7.3 ввожу в lab7-2.asm. (рис. ??).

```
lab7-2.asm [---] 11 L: [ 1+15 16/ 50] *(323 /17720) 0010 0x00A
%include "in_out.asm"
section .data
    msg1 db "Введите В: ",0h
    msg2 db "Наибольшее число: ",0h
    A dd '10'
    C dd '10'
section .bss
    max resb 10
    B resb 10
section .text
global _start
_start:
; ----- Вывод сообщения 'Введите В: '
mov eax,msg1
call sprint
; ----- Ввод 'В'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование 'В' из символа в число
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
```

Ввод текста программы из листинга 7.3

Создаю исполняемый файл и проверьте его работу. (рис. ??).

```
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ ld -m elf_i386 lab7-2.o -o lab7-2
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ ./lab7-2
Введите В: 53
Наибольшее число: 53
```

Проверка работы файла

Всё работает хорошо.

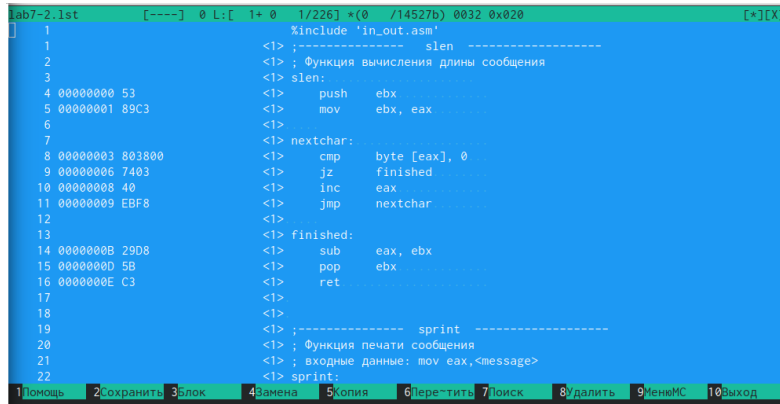
4.2 Изучение структуры файлы листинга

Создаю файл листинга для программы из файла lab7-2.asm. (рис. ??).

```
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
```

Создание файла листинга

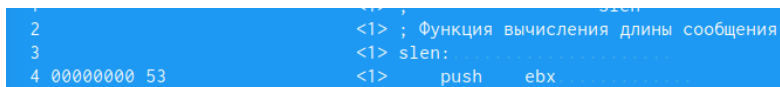
Открываю файл листинга lab7-2.lst с помощью текстового редактора и внимательно изучаю его формат и содержимое. (рис. ??).



```
lab7-2.lst  [----]  0 L: [ 1+ 0 1/226] *(0 /14527b) 0032 0x020 [*)(X]
1      %include 'in_out.asm'
2      <I> ----- slen -----
3      <I> ; Функция вычисления длины сообщения
4      <I> slen:
5      00000000 53      <I> push    ebx
6      00000001 89C3    <I> mov     ebx, eax
7
8      00000003 803800    <I> nextchar:
9      00000006 7403    <I> cmp     byte [eax], 0
10     00000008 40      <I> jz      finished
11     00000009 EBF8    <I> inc     eax
12     0000000A 7503    <I> jmp     nextchar
13     0000000B 29D8    <I> finished:
14     0000000D 5B      <I> sub     eax, ebx
15     0000000E C3      <I> pop     ebx
16     0000000F C3      <I> ret
17
18     00000010 7403    <I> ----- sprint -----
19     00000013 89C3    <I> ; Функция печати сообщения
20     00000016 89C3    <I> ; входные данные: mov eax, <message>
21     00000019 89C3    <I> sprint:
22     0000001C 89C3    <I> push    ebx
```

Изучение файла листинга

В представленных трех строчках содержатся следующие данные: (рис. ??).



```
2      <I> ; Функция вычисления длины сообщения
3      <I> slen:
4      00000000 53      <I> push    ebx
```

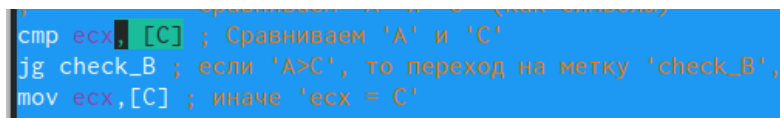
Выбранные строки файла

“2” - номер строки кода, “; Функция вычисления длинны сообщения” - комментарий к коду, не имеет адреса и машинного кода.

“3” - номер строки кода, “slen” - название функции, не имеет адреса и машинного кода.

“4” - номер строки кода, “00000000” - адрес строки, “53” - машинный код, “push ebx” - исходный текст программы, инструкция “push” помещает операнд “ebx” в стек.

Открываю файл с программой lab7-2.asm и в выбранной мной инструкции с двумя операндами удаляю выделенный операнд. (рис. ??).



```
стр есх, [C] ; Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', то переход на метку 'check_B',
mov есх, [C] ; иначе 'есх = C'
```

Удаление выделенного операнда

Выполняю трансляцию с получением файла листинга. (рис. ??).

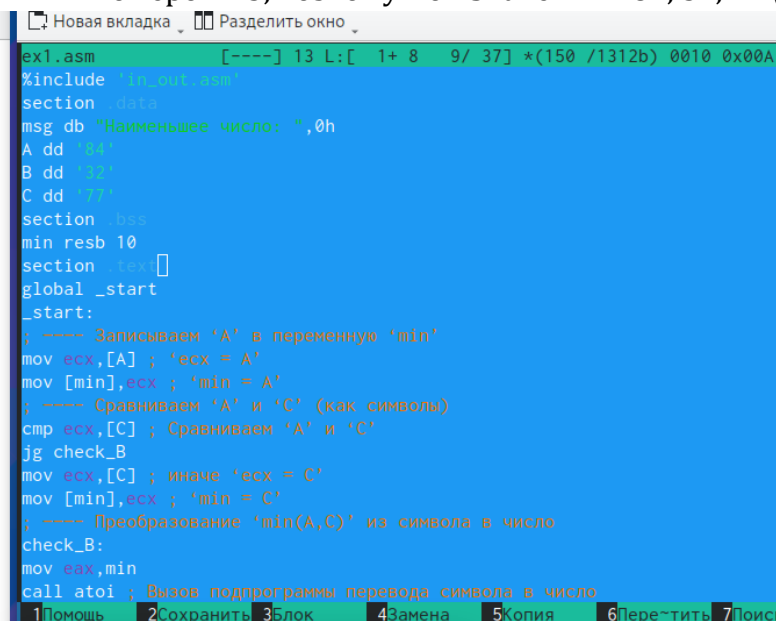
```
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:29: error: invalid combination of opcode and operands
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $
```

Получение файла листинга

На выходе я не получаю ни одного файла из-за ошибки: инструкция `mov` (единственная в коде содержит два операнда) не может работать, имея только один операнд, из-за чего нарушается работа кода.

4.3 Задания для самостоятельной работы

1. Пишу программу нахождения наименьшей из 3 целочисленных переменных `a`, `b` и `c`. Значения переменных выбираю из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Мой вариант под номером 13, поэтому мои значения - 84, 32, 77 (рис. ??).



```
ex1.asm [----] 13 L: [ 1+ 8 9/ 37] *(150 /1312b) 0010 0x00A
%include "in_out.asm"
section .data
msg db "Наименьшее число: ",0h
A dd '84'
B dd '32'
C dd '77'
section .bss
min resb 10
section .text
global _start
_start:
; ----- Записываем 'A' в переменную 'min'
mov ecx,[A] ; 'ecx = A'
mov [min],ecx ; 'min = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B
mov ecx,[C] ; иначе 'ecx = C'
mov [min],ecx ; 'min = C'
; ----- Преобразование 'min(A,C)' из символа в число
check_B:
mov eax,min
call atoi ; Вызов подпрограммы перевода символа в число
1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перетянуть 7Поиск
```

Написание программы

Создаю исполняемый файл и проверяю его работу, подставляя необходимые значения. (рис. ??).

```
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ nasm -f elf ex1.asm
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ ld -m elf_i386 ex1.o -o ex1
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ ./ex1
Наименьшее число: 32
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $
```

Запуск файла

2. Пишу программу, которая для введенных с клавиатуры значений `x` и `a` вычисляет значение и выводит результат вычислений заданной для моего варианта функции $f(x)$:

`a - 7`, если `a >= 7`

$a * x$, если $a < 7$

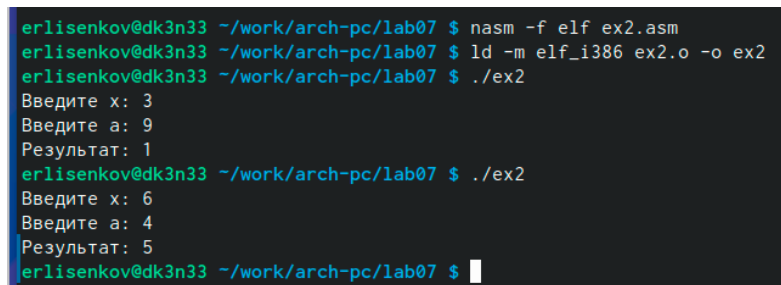
(рис. ??).



```
ex2.asm [-M--] 9 L: [ 1+13 14/ 42] *(244 / 554b) 0010 0x00a [*][X]
%include "stdio.asm"
section .data
vvdex: db "Введите x: ",0
vvoda: db "Введите a: ",0
vivod: db "Результат: ",0
section .bss
x: resb 80
a: resb 80
section .text
global _start
_start:
mov ecx, vvdex
call sprint
mov ecx, 1
mov edx, 80
call sread
mov ecx, a
call atoi
cmp ecx, 7
jg _functionx
mov ecx, vvoda
call sprint
mov ecx, a
```

Написание программы

Создаю исполняемый файл и проверяю его работу для значений x и a соответственно: (3;9), (6;4). (рис. ??).



```
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ nasm -f elf ex2.asm
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ ld -m elf_i386 ex2.o -o ex2
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ ./ex2
Введите x: 3
Введите a: 9
Результат: 1
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $ ./ex2
Введите x: 6
Введите a: 4
Результат: 5
erlisenkov@dk3n33 ~/work/arch-pc/lab07 $
```

Запуск файла

5 Выводы

Благодаря этой лабораторной работе, я закрепил свои знания в работе в программирование ветвлений.

6 Список литературы

1. Лабораторная работа №7. Команды безусловного и условного переходов в Nasm. Программирование ветвлений.