

Отчёт по лабораторной работе №8

Уткина Алина Дмитриевна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация переходов в NASM	6
2.2	Изучение структуры файлов листинга	10
2.3	Самостоятельная работа	11
3	Выводы	14

Список иллюстраций

2.1	Листинг 8.1. Программа с использованием инструкции jmp	6
2.2	Результат работы программы lab8-1.asm	7
2.3	Листинг 8.2. Измененный текст программы lab8-1.asm	7
2.4	Результат измененной программы	8
2.5	Другой вариант программы lab8-1.asm	8
2.6	Результат работы третьей программы	8
2.7	Листинг 8.3. Программа определения максимального из 3 чисел .	9
2.8	Результат работы программы lab8-2.asm	9
2.9	Создание файла листинга	10
2.10	Формат файла листинга	10
2.11	Область изменения программы	11
2.12	Результат трансляции файла с ошибкой	11
2.13	Запись в файле листинга с указанием ошибки	11
2.14	программа для первого задания самостоятельной работы	12
2.15	Результат выполнения первой программы	12
2.16	Функция для выполнения второй программы	12
2.17	Вторая программа	13
2.18	Результат выполнения второй программы	13

Список таблиц

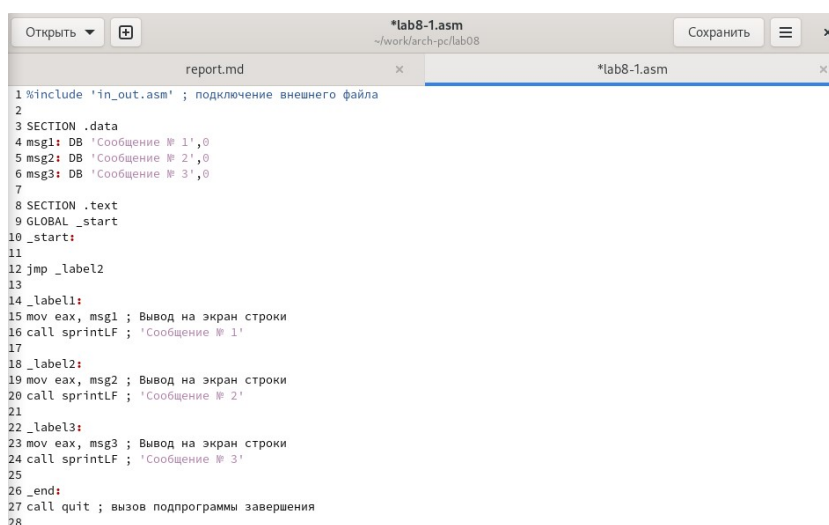
1 Цель работы

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

2 Выполнение лабораторной работы

2.1 Реализация переходов в NASM

Создадим каталог для программ лабораторной работы № 8, перейдем в него и создадим файл lab8-1.asm. Инструкция jmp в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции jmp. Введем в файл lab8-1.asm текст программы из листинга 8.1 (рис. 2.1).



```
1 %include 'in_out.asm' ; подключение внешнего файла
2
3 SECTION .data
4 msg1: DB 'Сообщение № 1',0
5 msg2: DB 'Сообщение № 2',0
6 msg3: DB 'Сообщение № 3',0
7
8 SECTION .text
9 GLOBAL _start
10 _start:
11
12 jmp _label2
13
14 _label1:
15 mov eax, msg1 ; Вывод на экран строки
16 call sprintf ; 'Сообщение № 1'
17
18 _label2:
19 mov eax, msg2 ; Вывод на экран строки
20 call sprintf ; 'Сообщение № 2'
21
22 _label3:
23 mov eax, msg3 ; Вывод на экран строки
24 call sprintf ; 'Сообщение № 3'
25
26 _end:
27 call quit ; вызов подпрограммы завершения
28
```

Рис. 2.1: Листинг 8.1. Программа с использованием инструкции jmp

Создадим исполняемый файл и запустим его. В результате работы данной программы будут выведены строки “Сообщение №2” и “Сообщение №3” (рис. 2.2).

```

[adutkina@fedora lab08]$ touch lab8-1.asm
[adutkina@fedora lab08]$ gedit lab8-1.asm
[adutkina@fedora lab08]$ nasm -f elf lab8-1.asm
[adutkina@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[adutkina@fedora lab08]$ ./lab8-1
Сообщение № 2
Сообщение № 3
[adutkina@fedora lab08]$

```

Рис. 2.2: Результат работы программы lab8-1.asm

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

Инструкция `jmp` позволяет осуществлять переходы не только вперед но и назад. Изменим программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. Для этого в текст программы после вывода сообщения № 2 добавим инструкцию `jmp` с меткой `_label1` (т.е. переход к инструкциям вывода сообщения № 1) и после вывода сообщения № 1 добавим инструкцию `jmp` с меткой `_end` (т.е. переход к инструкции `call quit`). Изменим текст программы в соответствии с листингом 8.2 (рис. 2.3). При запуске программы выводится именно то, что нам нужно (рис. 2.4).

```

report.md x *lab8-1.asm x
1 %include 'in_out.asm' ; подключение внешнего файла
2
3 SECTION .data
4 msg1: DB 'Сообщение № 1',0
5 msg2: DB 'Сообщение № 2',0
6 msg3: DB 'Сообщение № 3',0
7
8 SECTION .text
9 GLOBAL _start
10 _start:
11
12 jmp _label2
13
14 _label1:
15 mov eax, msg1 ; Вывод на экран строки
16 call sprintf ; 'Сообщение № 1'
17
18 jmp _end
19
20 _label2:
21 mov eax, msg2 ; Вывод на экран строки
22 call sprintf ; 'Сообщение № 2'
23
24 jmp _label1
25
26 _label3:
27 mov eax, msg3 ; Вывод на экран строки
28 call sprintf ; 'Сообщение № 3'
29
30 _end:
31 call quit ; вызов подпрограммы завершения
32

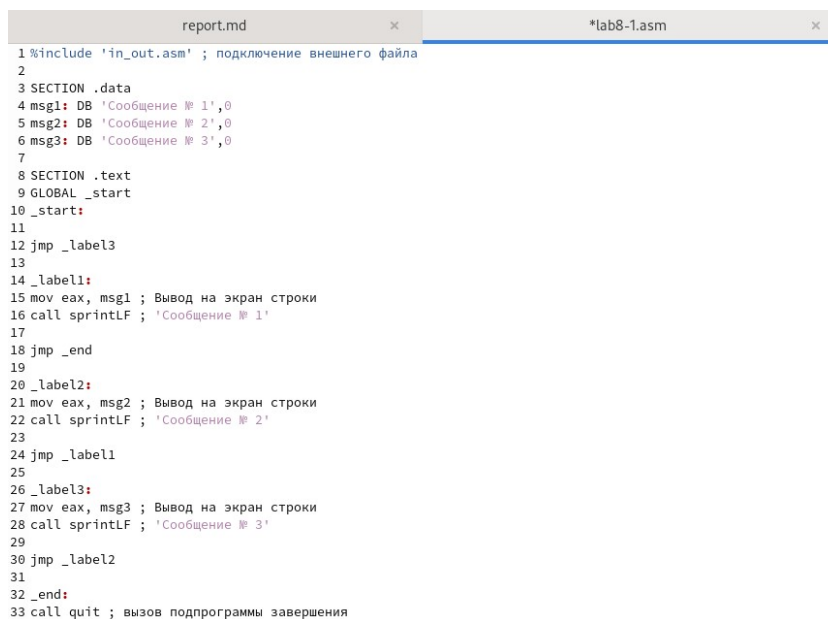
```

Рис. 2.3: Листинг 8.2. Измененный текст программы lab8-1.asm

```
[adutkina@fedora lab08]$ ./lab8-1
Сообщение № 2
Сообщение № 1
[adutkina@fedora lab08]$
```

Рис. 2.4: Результат измененной программы

Изменим текст программы добавив или изменив инструкции `jmp`, чтобы программа выводила сообщения в обратном порядке (рис. 2.5). Запустим исполняемый файл, чтобы проверить его работу (рис. 2.6).



```
report.md x *lab8-1.asm x
1 %include 'in_out.asm' ; подключение внешнего файла
2
3 SECTION .data
4 msg1: DB 'Сообщение № 1',0
5 msg2: DB 'Сообщение № 2',0
6 msg3: DB 'Сообщение № 3',0
7
8 SECTION .text
9 GLOBAL _start
10 _start:
11
12 jmp _label3
13
14 _label1:
15 mov eax, msg1 ; Вывод на экран строки
16 call sprintf ; 'Сообщение № 1'
17
18 jmp _end
19
20 _label2:
21 mov eax, msg2 ; Вывод на экран строки
22 call sprintf ; 'Сообщение № 2'
23
24 jmp _label1
25
26 _label3:
27 mov eax, msg3 ; Вывод на экран строки
28 call sprintf ; 'Сообщение № 3'
29
30 jmp _label2
31
32 _end:
33 call quit ; вызов подпрограммы завершения
```

Рис. 2.5: Другой вариант программы lab8-1.asm

```
[adutkina@fedora lab08]$ ./lab8-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
[adutkina@fedora lab08]$
```

Рис. 2.6: Результат работы третьей программы

Использование инструкции `jmp` приводит к переходу в любом случае. Однако, часто при написании программ необходимо использовать условные переходы, т.е. переход должен происходить если выполнено какое-либо условие. В качестве примера 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. Создадим файл lab8-2.asm в каталоге ~/work/arch-pc/lab08. Внимательно изучим текст программы из листинга 8.3 и введем его в lab8-2.asm (рис. 2.7). Создадим исполняемый файл и запустим его (рис. 2.8)

```

1 %include 'in_out.asm'
2 section .data
3 msg1 db 'Введите B: ',0h
4 msg2 db "Наибольшее число: ",0h
5 A dd '20'
6 C dd '50'
7 section .bss
8 max resb 10
9 B resb 10
10 section .text
11 global _start
12 _start:
13 ; ----- Вывод сообщения 'Введите B: '
14 mov eax,msg1
15 call sprint
16 ; ----- Ввод 'B'
17 mov ecx,B
18 mov edx,10
19 call sread
20 ; ----- Преобразование 'B' из символа в число
21 mov eax,B
22 call atoi ; Вызов подпрограммы перевода символа в число
23 mov [B],eax ; запись преобразованного числа в 'B'
24 ; ----- Записываем 'A' в переменную 'max'
25 mov ecx,[A] ; 'ecx = A'
26 mov [max],ecx ; 'max = A'
27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 jg check_B ; если 'A>C', то переход на метку 'check_B',
30 mov ecx,[C] ; иначе 'ecx = C'
31 mov [max],ecx ; 'max = C'
32 ; ----- Преобразование 'max(A,C)' из символа в число
33 check_B:
34 mov eax,max
35 call atoi ; Вызов подпрограммы перевода символа в число
36 mov [max],eax ; запись преобразованного числа в 'max'
37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
38 mov ecx,[max]
39 cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
40 jg fin ; если 'max(A,C)>B', то переход на 'fin',
41 mov ecx,[B] ; иначе 'ecx = B'
42 mov [max],ecx
43 ; ----- Вывод результата
44 fin:
45 mov eax,msg2
46 call sprint ; Вывод сообщения 'Наибольшее число: '
47 mov eax,[max]
48 call iprintf ; Вывод 'max(A,B,C)'
49 call quit ; Выход
50

```

Рис. 2.7: Листинг 8.3. Программа определения максимального из 3 чисел

```

[adutkina@fedora lab08]$ ./lab8-2
Введите B: 5
Наибольшее число: 50
[adutkina@fedora lab08]$ ./lab8-2
Введите B: 60
Наибольшее число: 60
[adutkina@fedora lab08]$

```

Рис. 2.8: Результат работы программы lab8-2.asm

Следует заметить, что в данном примере переменные А и С сравниваются как символы, а переменная В и максимум из А и С как числа (для этого используется функция atoi преобразования символа в число). Это сделано для демонстрации

того, как сравниваются данные. Данную программу можно упростить и сравнивать все 3 переменные как символы (т.е. не использовать функцию atoi). Однако если переменные преобразовать из символов в числа, над ними можно корректно проводить арифметические операции.

2.2 Изучение структуры файлов листинга

Обычно nasm создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ -l и задав имя файла листинга в командной строке. Создадим файл листинга для программы из файла lab8-2.asm (рис. 2.9).

```
[adutkina@fedora lab08]$ nasm -f elf -l lab8-2.lst lab8-2.asm
[adutkina@fedora lab08]$ ls
in_out.asm  lab8-1.asm  lab8-2      lab8-2.lst
lab8-1      lab8-1.o    lab8-2.asm  lab8-2.o
```

Рис. 2.9: Создание файла листинга

Откроем файл листинга lab8-2.lst с помощью текстового редактора, ознакомимся с его форматом и содержимым. Рассмотрим содержимое трёх строк (192-194) файла листинга (рис. 2.10).

```
192    17                                ; ----- Вывод сообщения 'Введите B: '
193    18 000000E8 B8[00000000]        mov eax,msg1
194    19 000000ED E81DFFFFFF          call sprint
```

Рис. 2.10: Формат файла листинга

Можно заметить, что номера строки файла листинга не совпадают с номерами строк исходного текста программ. Это связано с тем, что перед самой программой в листинге присутствует информация о функциях, используемых из подключаемого файла. В первой из трех рассматриваемых строк нет адреса и машинного кода, так как исходный текст программы - это только комментарий, а значит и машинный код не генерируется. В следующих двух строчках исходного кода содержатся команды, поэтому у них есть и адрес - смещение машинного кода от

начала сегмента, и машинный код - ассемблированная исходная строка, инструкция на машинном языке, вызывающая прерывание ядра: 000000EB и B8[00000000] - адрес и код соответственно для команды mov, 000000ED и E81DFFFFFF - для команды call sprint.

Откроем файл с программой lab8-2.asm и в инструкции mov (строка 22) с двумя операндами (ecx, B) удалим второй операнд (рис. 2.11). Выполним трансляцию с получением файла листинга (рис. 2.12). В результате работы выдается ошибка с указанием на номер неправильной строки (22). Создается файл lab8-2.lst, в котором добавляется дополнительная строка (с тем же номером) отмеченная звездочками с указанием проблемы (рис. 2.13).

```
21 ; ----- Ввод 'B'
22 mov ecx,B
23 mov edx,10
24 call sread
```

Рис. 2.11: Область изменения программы

```
[adutkina@fedora lab08]$ nasm -f elf -l lab8-2.lst lab8-2.asm
lab8-2.asm:22: error: invalid combination of opcode and operands
[adutkina@fedora lab08]$ ls
in_out.asm lab8-1 lab8-1.asm lab8-1.o lab8-2 lab8-2.asm lab8-2.lst
```

Рис. 2.12: Результат трансляции файла с ошибкой

```
196 21 ; ----- Ввод 'B'
197 22 mov ecx
198 22 ***** error: invalid combination of opcode and operands
199 23 000000F2 BA0A000000 mov edx,10
200 24 000000F7 E847FFFFFF call sread
```

Рис. 2.13: Запись в файле листинга с указанием ошибки

2.3 Самостоятельная работа

Напишем программу нахождения наименьшей из 3 целочисленных переменных a, b и c, где значения переменных равны 32, 6 и 54 соответственно (рис. 2.14). Создадим исполняемый файл и проверим его работу (рис. 2.15).

```

1 %include 'in_out.asm'
2 section .data
3 msg1 db 'Введите B: ',0h
4 msg2 db "Наибольшее число: ",0h
5 A dd '32'
6 B dd '6'
7 C dd '54'
8 section .bss
9 max resb 10
10 section .text
11 global _start
12 _start:
13 ; ----- Преобразование 'B' из символа в число
14 mov eax,B
15 call atoi ; Вызов подпрограммы перевода символа в число
16 mov [B],eax ; запись преобразованного числа в 'B'
17 ; ----- Записываем 'A' в переменную 'max'
18 mov ecx,[A] ; 'ecx = A'
19 mov [max],ecx ; 'max = A'
20 ; ----- Сравниваем 'A' и 'C' (как символы)
21 cmp ecx,[C] ; Сравниваем 'A' и 'C'
22 jg check_B ; если 'A>C', то переход на метку 'check_B',
23 mov ecx,[C] ; иначе 'ecx = C'
24 mov [max],ecx ; 'max = C'
25 ; ----- Преобразование 'max(A,C)' из символа в число
26 check_B:
27 mov eax,max
28 call atoi ; Вызов подпрограммы перевода символа в число
29 mov [max],eax ; запись преобразованного числа в 'max'
30 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
31 mov ecx,[max]
32 cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
33 jg fin ; если 'max(A,C)>B', то переход на 'fin',
34 mov ecx,[B] ; иначе 'ecx = B'
35 mov [max],ecx
36 ; ----- Вывод результата
37 fin:
38 mov eax, msg2
39 call sprint ; Вывод сообщения 'Наибольшее число: '
40 mov eax,[max]
41 call iprintLF ; Вывод 'max(A,B,C)'
42 call quit ; Выход
43

```

Рис. 2.14: программа для первого задания самостоятельной работы

```

[adutkina@fedora report]$ cd
[adutkina@fedora ~]$ cd work/arch-pc/lab08/
[adutkina@fedora lab08]$ ./var8-1
Наибольшее число: 54

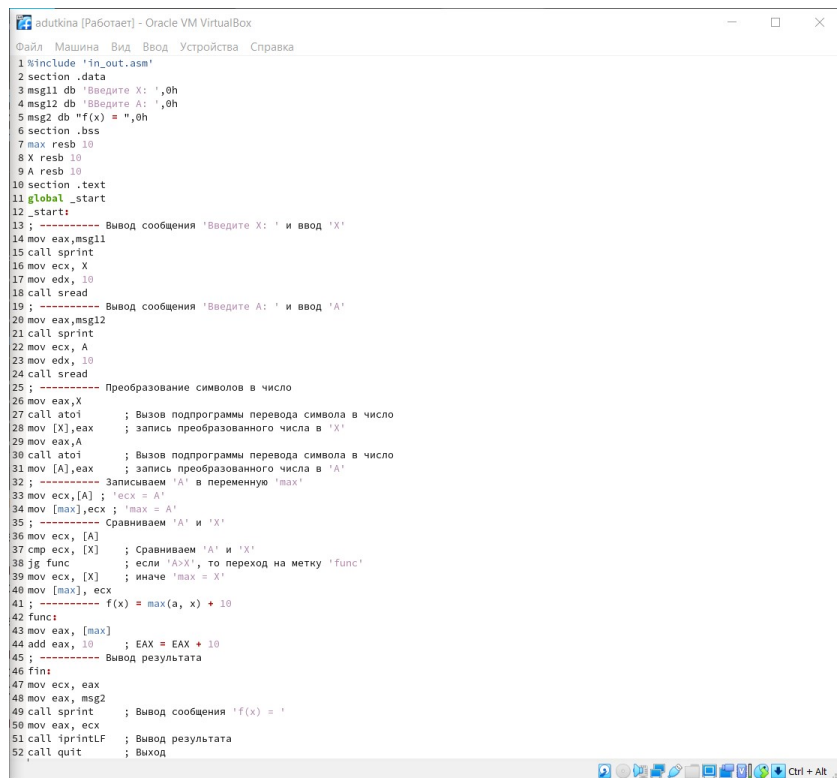
```

Рис. 2.15: Результат выполнения первой программы

Напишем программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции №15 (рис. 2.16) и выводит результат вычислений (рис. 2.17). Создадим исполняемый файл и проверим его работу для значений x и a равных 2, 3 соответственно для первого теста и 4, 2 - для второго (рис. 2.18).

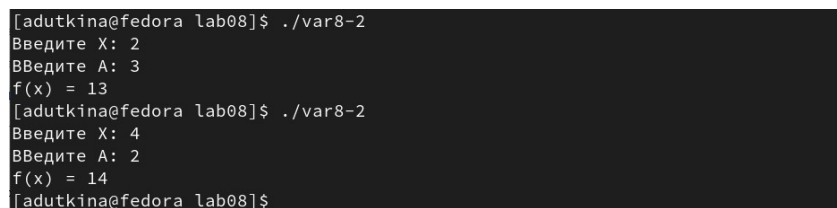
$$15 \quad \begin{cases} a + 10, & x < a \\ x + 10, & x \geq a \end{cases}$$

Рис. 2.16: Функция для выполнения второй программы



```
adutkina [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
1 %include 'in_out.asm'
2 section .data
3 msg11 db 'Введите X: ',0h
4 msg12 db 'Введите A: ',0h
5 msg2 db "f(x) = ",0h
6 section .bss
7 max resb 10
8 X resb 10
9 A resb 10
10 section .text
11 global _start
12 _start:
13 ; ----- Вывод сообщения 'Введите X: ' и ввод 'X'
14 mov eax,msg11
15 call sprint
16 mov ecx, X
17 mov edx, 10
18 call sread
19 ; ----- Вывод сообщения 'Введите A: ' и ввод 'A'
20 mov eax,msg12
21 call sprint
22 mov ecx, A
23 mov edx, 10
24 call sread
25 ; ----- Преобразование символов в число
26 mov eax,X
27 call atoi ; Вызов подпрограммы перевода символа в число
28 mov [X],eax ; запись преобразованного числа в 'X'
29 mov eax,A
30 call atoi ; Вызов подпрограммы перевода символа в число
31 mov [A],eax ; запись преобразованного числа в 'A'
32 ; ----- Записываем 'A' в переменную 'max'
33 mov ecx,[A] ; 'ecx = A'
34 mov [max],ecx ; 'max = A'
35 ; ----- Сравниваем 'A' и 'X'
36 mov ecx, [A]
37 cmp ecx, [X] ; Сравниваем 'A' и 'X'
38 jg func ; если 'A>X', то переход на метку 'func'
39 mov ecx, [X] ; иначе 'max = X'
40 mov [max], ecx
41 ; ----- f(x) = max(a, x) + 10
42 func:
43 mov eax, [max]
44 add eax, 10 ; EAX = EAX + 10
45 ; ----- Вывод результата
46 fin:
47 mov ecx, eax
48 mov eax, msg2
49 call sprint ; Вывод сообщения 'f(x) = '
50 mov eax, ecx
51 call iprintfLF ; Вывод результата
52 call quit ; Выход
```

Рис. 2.17: Вторая программа



```
[adutkina@fedora lab08]$ ./var8-2
Введите X: 2
Введите A: 3
f(x) = 13
[adutkina@fedora lab08]$ ./var8-2
Введите X: 4
Введите A: 2
f(x) = 14
[adutkina@fedora lab08]$
```

Рис. 2.18: Результат выполнения второй программы

3 Выводы

В ходе данной работы были изучены команды условного, безусловного переходов и назначение, структура файла листинга, приобретены навыки написания программ с использованием переходов.