

Лабораторная работа №8

**Команды безусловного и условного переходов в Nasm.
Программирование ветвлений.**

Татьяна Алексеевна Коннова, НПИбд-01-22

Содержание

1	Цель работы	4
1.1	Задание	4
1.2	Выполнение лабораторной работы	4
1.3	Изучение структуры файлы листинга	11
1.4	Самостоятельная работа	13
2	Выводы	18

Список иллюстраций

1.1	lab8_1	5
1.2	lab8_1	6
1.3	Change	6
1.4	First change lab8_1	6
1.5	End change	8
1.6	Output 3 2 1	9
1.7	lab8_2	10
1.8	output lab8_2	11
1.9	Error in listing	12
1.10	Номер 1 Самостоятельной работы	14
1.11	Вывод программы номера 1	14
1.12	Программа 2 Самостоятельной работы	16
1.13	Вывод программы номер 2 Самостоятельной работы	17

1 Цель работы

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

1.1 Задание

Программирование ветвлений

1.2 Выполнение лабораторной работы

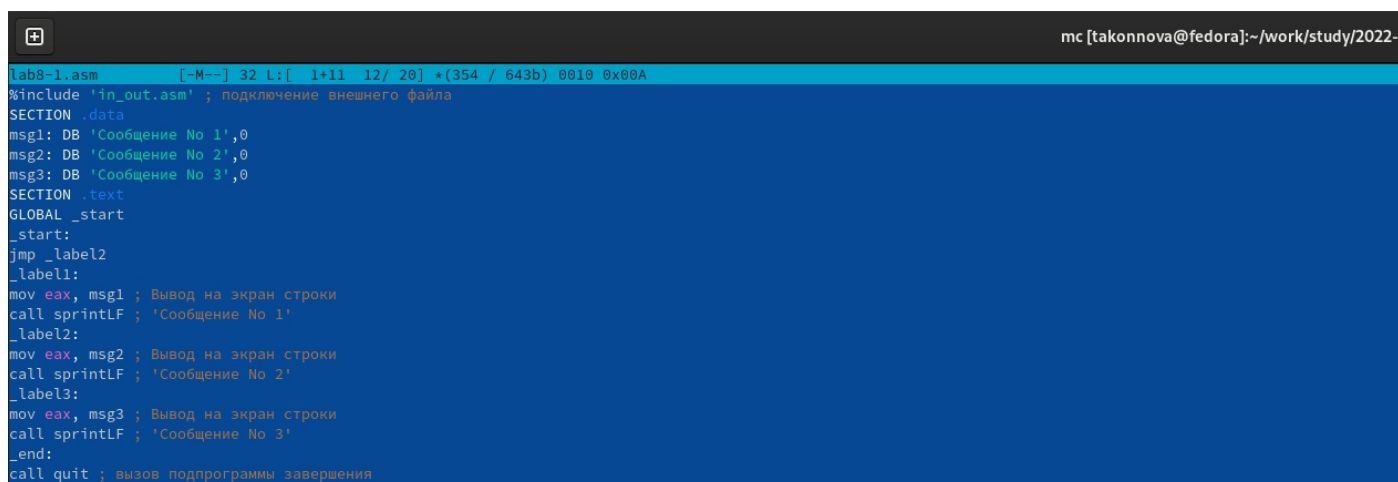
1. Создаем каталог для программам лабораторной работы N 8, переходим в него и создайте файл lab8-1.asm:

```
mkdir ~/work/arch-pc/lab08
```

```
cd ~/work/arch-pc/lab08
```

```
touch lab8-1.asm
```

2. 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. 1.1)



```
lab8-1.asm      [-M--] 32 L:[ 1+11 12/ 20] *(354 / 643b) 0010 0x00A
mc [takonnova@fedora]:~/work/study/2022-

%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение No 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение No 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение No 3'
_end:
call quit ; вызов подпрограммы завершения
```

Рис. 1.1: lab8_1

Создаем исполняемый файл и запускаем его. Результат работы данной программы будет следующим:

./lab8-1

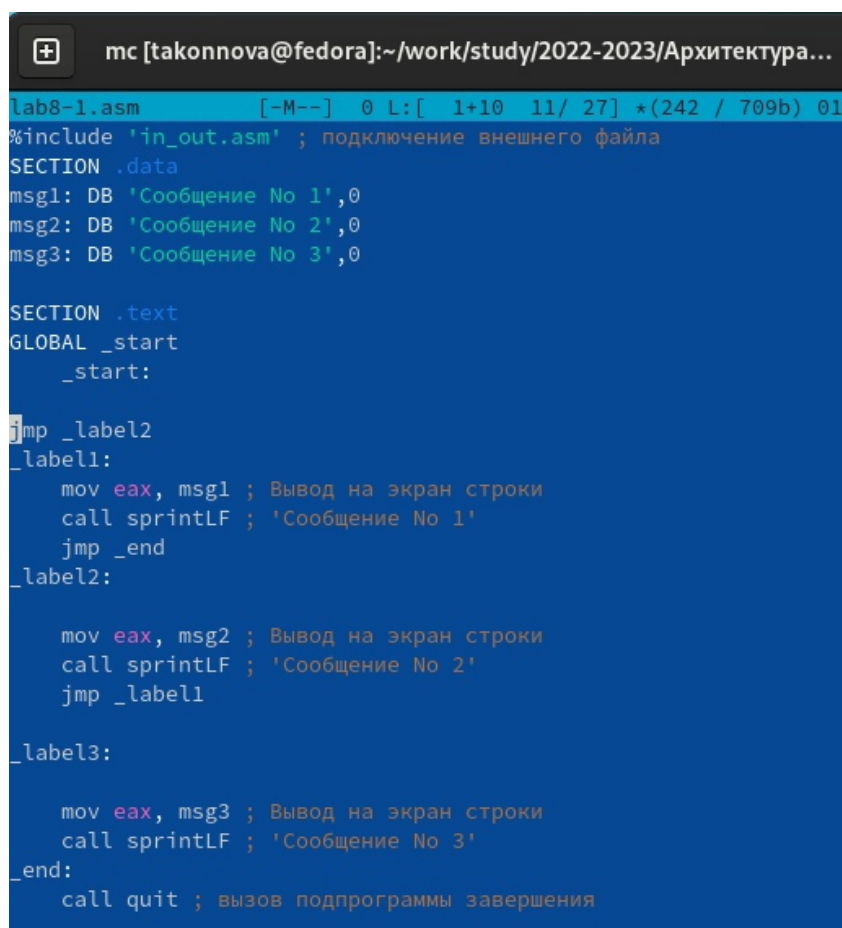
Сообщение No 2

Сообщение No 3

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 No 2', потом 'Сообщение No 1' и завершала работу. Для этого в текст программы после вывода сообщения No 2 добавим инструкцию `jmp` с меткой `_label1` (т.е. переход к инструкциям вывода сообщения No 1) и после вывода сообщения No 1 добавим инструкцию `jmp` с меткой `_end` (т.е. переход к инструкции `call quit`). Изменим текст программы (рис. 1.2) (рис. 1.3) (рис. 1.4)

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

Рис. 1.2: lab8_1



```
mc [takonnova@fedora]:~/work/study/2022-2023/Архитектура...
lab8-1.asm [-M--] 0 L:[ 1+10 11/ 27] *(242 / 709b) 01
#include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label2
_label1:
    mov eax, msg1 ; Вывод на экран строки
    call sprintf ; 'Сообщение No 1'
    jmp _end
_label2:

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

_label3:

    mov eax, msg3 ; Вывод на экран строки
    call sprintf ; 'Сообщение No 3'
_end:
    call quit ; вызов подпрограммы завершения
```

Рис. 1.3: Change

```
[takonnova@fedora lab08]$ nasm -f elf lab8-1.asm
[takonnova@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[takonnova@fedora lab08]$ ./lab8-1
Сообщение No 2
Сообщение No 1
```

Рис. 1.4: First change lab8_1

Создаем исполняемый файл и проверяем его работу.

Изменяем текст программы добавив или изменив инструкции `jmp`, чтобы вывод программы был следующим: (рис. 1.5) (рис. 1.6) ./lab8-1

Сообщение No 3

Сообщение No 2

Сообщение No 1

```
mc [takonnova@fedora]:~/work/study/2022-2
lab8-1.asm [----] 11 L:[ 1+10 11/ 28]
#include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0

SECTION .text
GLOBAL _start
_start:
    jmp _label3
_label1:
    mov eax, msg1 ; Вывод на экран строки
    call sprintf ; 'Сообщение No 1'
    jmp _end
_label2:
    mov eax, msg2 ; Вывод на экран строки
    call sprintf ; 'Сообщение No 2'
    jmp _label1
_label3:
    mov eax, msg3 ; Вывод на экран строки
    call sprintf ; 'Сообщение No 3'
    jmp _label2
_end:
    call quit ; вызов подпрограммы завершения
```

Рис. 1.5: End change


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

Рис. 1.6: Output 3 2 1

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: A, B и C. Значения для A и C задаются в программе, значение B вводится с клавиатуры. Создаем файл `lab8-2.asm` в каталоге

`~/work/arch-pc/lab08`

Внимательно изучаем текст программы из листинга 8.3 и введем в `lab8-2.asm`.
(рис. 1.7) (рис. 1.8)

Создаем исполняемый файл и проверяем его работу для разных значений B.

```
mc [takonnova@fedora]:~/work/study/2022-2023/Архитектура
lab8-2.asm [----] 13 L:[ 1+10 11/ 49] *(197 /1743b)
#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:
; ----- Вывод сообщения 'Введите B: '
mov eax,msg1
call sprint
; ----- Ввод 'B'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование 'B' из символа в число
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'B'
; ----- Записываем 'A' в переменную 'max'
mov ecx,[A] ; 'ecx = A'
mov [max],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', то переход на метку 'check_B',
mov ecx,[C] ; иначе 'ecx = C'
mov [max],ecx ; 'max = C'
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
mov eax,max
call atoi ; Вызов подпрограммы перевода символа в число
mov [max],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx,[max]
cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [max],ecx
; ----- Вывод результата
fin:
mov eax, msg2
call sprint ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintLF ; Вывод 'max(A,B,C)'
call quit ; Выход
```

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8Удалить

Рис. 1.7: lab8_2

```
[takonnova@fedora lab08]$ nasm -f elf lab8-2.asm
[takonnova@fedora lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[takonnova@fedora lab08]$ ./lab8-2
Введите В: 6
Наибольшее число: 50
```

Рис. 1.8: output lab8_2

Обращаем внимание, 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 числа, над ними можно корректно проводить арифметические операции.

1.3 Изучение структуры файлы 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. Создаем файл листинга для программы из файла `lab8-2.asm`

```
nasm -f elf -l lab8-2.lst lab8-2.asm
```

Откроем файл листинга `lab8-2.lst` с помощью любого текстового редактора, например `mcedit`:

```
mcedit lab8-2.lst
```

Внимательно ознакомимся с его форматом и содержимым. Подробно объясним содержимое трёх строк файла листинга по выбору.

строка 51

51 - номер строки

00000033 - адрес

B80A000000 - машинный код
mov eax, 0AH - код программы

строка 37

37 - номер строки
00000135 - адрес
E862FFFFFF - машинный код
call atoi- код программы

строка 53

53 - номер строки
00000039 - адрес
89E0 - машинный код
mov eax, esp - код программы

Откроем файл с программой lab8-2.asm и в любой инструкции с двумя операндами удалить один операнд. Выполним трансляцию с получением файла листинга:

```
nasm -f elf -l lab8-2.lst lab8-2.asm
```

Какие выходные файлы создаются в этом случае? Что добавляется в листинге?

Ответ: после удаления операнда появилась ошибка и файлы не формировались, а в листинге ошибка.(рис. 1.9)

```

/home/takonnova/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab08/lab8-2: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, not stripped
0804a035 d A
0804a090c t atoi
0804a090ca t atoi.finished
0804a090ac t atoi.multiplyLoop
0804a090d6 t atoi.restore
0804a04a b B
0804a03d B __bss_start
0804a039 d C
0804a0130 t check_B
0804a005d t divideLoop
0804a03d D _edata
0804a054 B _end
0804a0159 t fin
0804a000b t finished
0804a054 t iprint
0804a0086 t iprintLF
0804a040 b max
0804a000 d msg1
0804a013 d msg2
0804a003 t nextchar
0804a0073 t printLoop
0804a00db t quit
0804a000 t slen
0804a00f t sprint
0804a002d t sprintLF
0804a043 t sread
0804a00e T _start
```

Рис. 1.9: Error in listing

1.4 Самостоятельная работа

1. Напишите программу нахождения наименьшей из 3 целочисленных переменных a , b и c . Значения переменных выбрать из табл. 8.5 в соответствии с вариантом, полученным при выполнении лабораторной работы No 7. Создайте исполняемый файл и проверьте его работу.
2. Напишите программу, которая для введенных с клавиатуры значений x and a вычисляет значение заданной функции и выводит результат вычислений. Вид функции выбрать из таблицы 8.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы No 7. Создайте исполняемый файл и проверьте его работу для значений x and a

Мой вариант = 15. Напишем программу нахождения наибольшего из переменных a , b , c то есть 32, 6 и 54.

Создадим исполняемый файл и проверим его работу. (рис. 1.10) (рис. 1.11)

```

/home/takonnova/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab08/variantik8-1.asm
%include 'in_out.asm'
section .data
msg1 db 'Введите B: ',0h
msg2 db "Наибольшее число: ",0h
A dd '32'
B dd '6'
C dd '54'
section .bss
max resb 10
section .text
global _start
_start:
; ----- Преобразование 'B' из символа в число
mov eax,B
call atoi      ; Вызов подпрограммы перевода символа в число
mov [B],eax    ; запись преобразованного числа в 'B'
; ----- Записываем 'A' в переменную 'max'
mov ecx,[A]    ; 'ecx = A'
mov [max],ecx  ; 'max = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C]    ; Сравниваем 'A' и 'C'
jg check_B    ; если 'A>C', то переход на метку 'check_B',
mov ecx,[C]    ; иначе 'ecx = C'
mov [max],ecx  ; 'max = C'
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
mov eax,max
call atoi      ; Вызов подпрограммы перевода символа в число
mov [max],eax  ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx,[max]
cmp ecx,[B]    ; Сравниваем 'max(A,C)' и 'B'
jg fin        ; если 'max(A,C)>B', то переход на 'fin',
mov ecx,[B]    ; иначе 'ecx = B'
mov [max],ecx
; ----- Вывод результата
fin:
mov eax, msg2
call sprint    ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintLF  ; Вывод 'max(A,B,C)'
call quit     ; Выход

```

Рис. 1.10: Номер 1 Самостоятельной работы

```

[takonnova@fedora lab08]$ nasm -f elf variantik8-1.asm
[takonnova@fedora lab08]$ ld -m elf_i386 -o variantik8-1 variantik8-1.o
[takonnova@fedora lab08]$ ./variantik8-1
Наибольшее число: 54

```

Рис. 1.11: Вывод программы номера 1

Напишем вторую программу для введенных с клавиатуры значений x и a она

вычисляет значение заданной функции 15 и выводит результат вычислений. Создадим исполняемый файл и проверим его работу для значений 2, 3 для первого примера и 4, 2 - для вт. примера (рис. 1.12) (рис. 1.13)

```
mc [takonnova@fedora]:~/work/study/2022-2023/Архитектура..
variantik8-2.asm [----] 17 L:[ 1+ 4 5/ 52] *(120 /1574b)
%include 'in_ouo.asm'
section .data
msg11 db 'Введите X: ',0h
msg12 db 'Введите A: ',0h
msg2 db "f(x) = ",0h
section .bss
max resb 10
X resb 10
A resb 10
section .text
global _start
_start:
; ----- Вывод сообщения 'Введите X: ' и ввод 'X'
mov eax,msg11
call sprint
mov ecx, X
mov edx, 10
call sread
; ----- Вывод сообщения 'Введите A: ' и ввод 'A'
mov eax,msg12
call sprint
mov ecx, A
mov edx, 10
call sread
; ----- Преобразование символов в число
mov eax,X
call atoi <---->; Вызов подпрограммы перевода символа в число
mov [X],eax <-->; запись преобразованного числа в 'X'
mov eax,A
call atoi <---->; Вызов подпрограммы перевода символа в число
mov [A],eax <-->; запись преобразованного числа в 'A'
; ----- Записываем 'A' в переменную 'max'
mov ecx,[A] ; 'ecx = A'
mov [max],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'X'
mov ecx, [A]
cmp ecx, [X] <->; Сравниваем 'A' и 'X'
jg func><----->; если 'A>X', то переход на метку 'func'
mov ecx, [X]<-->; иначе 'max = X'
mov [max], ecx
; ----- f(x) = max(a, x) + 10.
func:
mov eax, [max]
add eax, 10<---->; EAX = EAX + 10
; ----- Вывод результата
fin:
mov ecx, eax
mov eax, msg2
call sprint <-->; Вывод сообщения 'f(x) = '
mov eax, ecx
call iprintLF <->; Вывод результата
call quit <---->; Выход
```

Рис. 1.12: Программа 2 Самостоятельной работы


```
[takonnova@fedora lab08]$ ./variantik8-2
Введите X: 2
Введите A: 3
f(x) = 13
[takonnova@fedora lab08]$ ./variantik8-2
Введите X: 4
Введите A: 2
f(x) = 14
```

Рис. 1.13: Вывод программы номер 2 Самостоятельной работы

2 Выводы

Мы научились командам условного и безусловного переходов, приобрели навыки написания программ с использованием переходов, познакомились с значением и структурой файла листинга.