

# **Отчёт по лабораторной работе №7**

**Дисциплина: Архитектура компьютера**

София Андреевна Кудякова

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задания</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
4.1	Реализация переходов в NASM . . . . .	8
4.2	Изучение структуры файлы листинга . . . . .	12
4.3	Выполнение заданий для самостоятельной работы . . . . .	14
<b>5</b>	<b>Выводы</b>	<b>22</b>
	<b>Список литературы</b>	<b>23</b>

## Список иллюстраций

4.1	Создание директории . . . . .	8
4.2	Создание файла . . . . .	8
4.3	Редактирование файла . . . . .	9
4.4	Запуск программы файла . . . . .	9
4.5	Редактирование файла . . . . .	10
4.6	Запуск программы файла . . . . .	10
4.7	Редактирование файла . . . . .	11
4.8	Исполнение файла . . . . .	11
4.9	Создание файла . . . . .	11
4.10	Ввод программы в файл . . . . .	12
4.11	Исполнение файла . . . . .	12
4.12	Создание файла листинга . . . . .	13
4.13	Открытие файла . . . . .	13
4.14	Выбанные строки . . . . .	13
4.15	Удаление операнда . . . . .	14
4.16	Трансляция с получением файла листинга . . . . .	14
4.17	Написание программы . . . . .	15
4.18	Исполнение файла . . . . .	15
4.19	Написание программы . . . . .	18
4.20	Запуск исполняемого файла . . . . .	19
4.21	Запуск исполняемого файла . . . . .	19

# 1 Цель работы

Цель данной лабораторной работы - изучить команды условного и безусловного переходов, научиться писать программы с использованием переходов, а также ознакомиться с назначением и структурой файла листинга.

## 2 Задания

1. Реализация переходов в NASM
2. Изучение структуры файлы листинга
3. Выполнение заданий 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: 1. Условный переход – выполнение или не выполнение перехода в определенную точку программы в зависимости от проверки условия. 2. Безусловный переход – выполнение передачи управления в определенную точку программы без каких-либо условий. Безусловный переход выполняется инструкцией `jmp` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление: `jmp Адрес перехода` может быть либо меткой, либо адресом области памяти, в которую предварительно помещен указатель перехода. Кроме того, в качестве операнда можно использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре. Для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов. Флаг – это бит, принимающий значение 1 («флаг установлен»), если выполнено некоторое условие, и значение 0 («флаг сброшен») в противном случае. Флаги работают независимо друг от друга, и лишь для удобства они помещены в единый регистр — регистр флагов, отражающий текущее состояние процессора. В следующей таблице указано положение битовых флагов в регистре флагов. Флаги состояния (биты 0, 2, 4, 6, 7 и 11) отражают результат выполнения арифметических инструкций, таких как `ADD`, `SUB`, `MUL`, `DIV`. Инструкция `cmp` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция `cmp` является

командой сравнения двух операндов и имеет такой же формат, как и команда вычитания: `cmp` , Команда `cmp`, так же как и команда вычитания, выполняет вычитание - , но результат вычитания никуда не записывается и единственным результатом команды сравнения является формирование флагов. Примеры:

`cmp ax, '4'` ; сравнение регистра `ax` с символом `4`

`cmp ax, 4` ; сравнение регистра `ax` со значением `4`

`cmp al, cl` ; сравнение регистров `al` и `cl`

`cmp [buf], ax` ; сравнение переменной `buf` с регистром `ax`

Команда условного перехода имеет вид: `j label` Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов. 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

Ввожу команду `mkdir`, с помощью которой создаю директорию, в которой буду создавать файлы. Перехожу в нее. (рис. 4.1).

```
sakudyakova@dk2n24 ~ $ mkdir ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/lab07
sakudyakova@dk2n24 ~ $ cd ~/work/study/2023-2024/"Архитектура компьютера"
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера $ cd arch-pc/lab07
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ █
```

Рис. 4.1: Создание директории

С помощью команды `touch` создаю файл `lab7-1.asm`. (рис. 4.2).

```
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ touch lab7-1.asm
```

Рис. 4.2: Создание файла

Открываю созданный файл в редакторе и вставляю в него программу с использованием инструкции `jmp`. (рис. 4.3).



```

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

```

Рис. 4.3: Редактирование файла

Создаю исполняемый файл и запускаю его. Программа работала корректно.  
(рис. 4.4).

```

sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ nasm -f elf lab7-1.asm
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 3
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ 

```

Рис. 4.4: Запуск программы файла

Инструкция `jmp` позволяет осуществлять переходы не только вперед но и назад. Изменяю 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`). (рис. 4.5).

```

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

```

Рис. 4.5: Редактирование файла

Создаю исполняемый файл и запускаю его. Программа отработала верно.(рис. 4.6).

```

sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ nasm -f elf lab7-1.asm
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 1
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $

```

Рис. 4.6: Запуск программы файла

Далее изменяю текст программы, добавив и изменив инструкции jmp. (рис. 4.7).

```

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

```

Рис. 4.7: Редактирование файла

Создаю исполняемый файл и запускаю его. Вывод верный, значит программа отработала корректно. (рис. 4.8).

```

sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ nasm -f elf lab7-1.asm
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ █

```

Рис. 4.8: Исполнение файла

Создаю новый файл lab7-2.asm в каталоге ~/work/arch-pc/lab06. (рис. 4.9).

```

sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ touch lab7-2.asm
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ █

```

Рис. 4.9: Создание файла

Ввожу в файл текст программа, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А,В и С. (рис. 4.10).

```

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'

```

Рис. 4.10: Ввод программы в файл

Создаю исполняемый файл и дважды запускаю его, чтобы проверить корректность работы программы. (рис. 4.11).

```

sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ nasm -f elf lab7-2.asm
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ./lab7-2
Введите B: 70
Наибольшее число: 70
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ./lab7-2
Введите B: 50
Наибольшее число: 50
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $

```

Рис. 4.11: Исполнение файла

Программа отработала верно

## 4.2 Изучение структуры файлы листинга

Создаю файл листинга для программы из файла lab7-2.asm. (рис. 4.12).

```
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $
```

Рис. 4.12: Создание файла листинга

Открываю файл листинга lab7-2.lst с помощью текстового редактора.(рис. 4.13).

```
GNU nano 7.2 lab7-2.lst
1 %include 'in_out.asm'
2 <1> ;----- slen -----
3 <1> ; Функция вычисления длины сообщения
4 <1> slen:
5 00000000 53 <1> push ebx
6 00000001 89C3 <1> mov ebx, eax
7 <1>
8 <1> nextchar:
9 00000003 803800 <1> cmp byte [eax], 0
10 00000006 7403 <1> jz finished
11 00000008 40 <1> inc eax
12 00000009 EBF8 <1> jmp nextchar
13 <1>
14 <1> finished:
15 0000000B 29D8 <1> sub eax, ebx
16 0000000D 5B <1> pop ebx
17 0000000E C3 <1> ret
18 <1>
19 <1> ;----- sprint -----
20 <1> ; Функция печати сообщения
21 <1> ; входные данные: mov eax,<message>
22 <1> sprint: _____
```

Рис. 4.13: Открытие файла

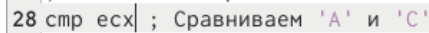
Для объяснения я выбрала строки 2; 3; 4. (рис. 4.14).

```
2 <1> ; Функция вычисления длины сообщения
3 <1> slen:
4 00000000 53 <1> push ebx
```

Рис. 4.14: Выбанные строки

Строка “2” - номер строки кода; ” ; Функция вычисление длины сообщения” - комментарий к коду, не имеет адреса и машинного кода. Строка “3” - номер строки кода; “slen” - название функции, не имеет адреса и машинного кода. Строка “4” - номер строки кода; “00000000” - адрес строки; “53” - машинный код; “push ebx” - исходный текст программы, push помещает операнд ebx в стек.

Открываю файл с программой lab7-2.asm и в инструкции с двумя операндами удаляю один операнд(рис. 4.15).



```
28 cmp eax, ecx ; Сравниваем 'А' и 'С'
```

Рис. 4.15: Удаление операнда

Выполняю трансляцию с получением файла листинга. (рис. 4.16).



```
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands
sakudyakova@dk2n24 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $
```

Рис. 4.16: Трансляция с получением файла листинга

В этом случае не создаются никакие выходные файлы из-за ошибки: инструкция `mov` не может работать, имея только один операнд. Поэтому код не работает.

## 4.3 Выполнение заданий для самостоятельной работы

1. Пишу программу нахождения наименьшей из 3 целочисленных переменных `a`, `b` и `c`. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Мой вариант - 14. Мои значения - 81,22,72 (рис. 4.17).

```

1 %include 'in_out.asm'
2 section .data
3 msg db "Наименьшее число: ",0h
4 A dd '81'
5 B dd '22'
6 C dd '72'
7 section .bss
8 min resb 10
9 section .text
10 global _start
11 _start:
12 ; ----- Записываем 'A' в переменную 'min'
13 mov ecx,[A] ; 'ecx = A'
14 mov [min],ecx ; 'min = A'
15 ; ----- Сравниваем 'A' и 'B' (как символы)
16 cmp ecx,[B] ; Сравниваем 'A' и 'B'
17 jg check_C ; если 'A<B', то переход на метку 'check_C',
18 mov ecx,[B] ; иначе 'ecx = B'
19 mov [min],ecx ; 'min = B'
20 ; ----- Преобразование 'min(A,B)' из символа в число
21 check_C:
22 mov eax,min
23 call atoi ; Вызов подпрограммы перевода символа в число
24 mov [min],eax ; запись преобразованного числа в min
25 ; ----- Сравниваем 'min(A,B)' и 'C' (как числа)
26 mov ecx,[min]
27 cmp ecx,[C] ; Сравниваем 'min(A,B)' и 'C'
28 jl fin ; если 'min(A,B)<C', то переход на 'fin',
29 mov ecx,[C] ; иначе 'ecx = C'
30 mov [min],ecx
31 ; ----- Вывод результата
32 fin:
33 mov eax, msg
34 call sprint ; Вывод сообщения 'Наименьшее число: '
35 mov eax,[min]
36 call iprintLF ; Вывод 'min(A,B,C)'
37 call quit ; Выход

```

Рис. 4.17: Написание программы

Создаю исполняемый файл и запускаю его. (рис. 4.18).

```

sakudyakova@dk6n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ nasm -f elf task1.asm
sakudyakova@dk6n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ld -m elf_i386 -o task1 task1.o
sakudyakova@dk6n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ./task1
Наименьшее число: 22

```

Рис. 4.18: Исполнение файла

Программа отработала верно.

**Листинг 4.2.1. Программа, которая определяет и выводит на экран наименьшую из 3 целочисленных переменных: A,B и C**

```

%include 'in_out.asm'

section .data
msg db "Наименьшее число: ",0h
A dd '81'
B dd '22'
C dd '72'

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 'B' (как символы)
cmp ecx,[B] ; Сравниваем 'A' и 'B'
jg check_C ; если 'A<B', то переход на метку 'check_C',
mov ecx,[B] ; иначе 'ecx = B'
mov [min],ecx ; 'min = B'
; ----- Преобразование 'min(A,B)' из символа в число
check_C:
mov eax,min
call atoi ; Вызов подпрограммы перевода символа в число
mov [min],eax ; запись преобразованного числа в min
; ----- Сравниваем 'min(A,B)' и 'C' (как числа)
mov ecx,[min]
cmp ecx,[C] ; Сравниваем 'min(A,B)' и 'C'
jl fin ; если 'min(A,B)<C', то переход на 'fin',

```



```

mov ecx,[C] ; иначе 'ecx = C'
mov [min],ecx
; ----- Вывод результата
fin:
mov eax, msg
call sprint ; Вывод сообщения 'Наименьшее число: '
mov eax,[min]
call iprintLF ; Вывод 'min(A,B,C)'
call quit ; Выход

```

2. Пишу программу, которая для введенных с клавиатуры значений  $x$  и  $a$  вычисляет значение заданной функции  $f(x)$  и выводит результат вычислений. Вид функции  $f(x)$  выбираю в соответствии с вариантом, полученным при выполнении лабораторной работы №6. Мой вариант 14, выбираю выражение для  $f(x)$  под номером 14. (рис. 4.19).

```

1 %include 'in_out.asm'
2 section .data
3 vvodx: db "Введите x: ",0
4 vvoda: db "Введите a: ",0
5 vivod: db "Результат: ",0
6
7 section .bss
8 x: resb 80
9 a: resb 80
10
11 section .text
12 global _start
13 _start:
14
15 mov eax,vvodx
16 call sprint
17 mov ecx,x
18 mov edx,80
19 call sread
20
21 mov eax,x
22 call atoi
23
24 mov edx,eax
25 mov eax,vvoda
26 call sprint
27 mov ecx,a
28 mov edx,80
29 call sread
30
31 mov eax,a
32 call atoi
33
34 cmp eax,edx
35 if < goto less

```

Рис. 4.19: Написание программы

Создаю исполняемый файл и запускаю его. Ввожу сначала значения 2 и 3 (рис. 4.20).

```
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ./task2
Введите x: 2
Введите a: 3
Результат: 10
```

Рис. 4.20: Запуск исполняемого файла

Затем ввожу значения 4 и 2. (рис. 4.21).

```
sakudyakova@dk1n22 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ./task2
Введите x: 4
Введите a: 2
Результат: 13
```

Рис. 4.21: Запуск исполняемого файла

Программа работает верно

#### Листинг 4.2.2. Программа для вычисления значения заданной функции

```
%include 'in_out.asm'

section .data
vvodb: 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 eax,vvodb
call sprint
mov ecx,x
```

```
mov edx,80
```

```
call sread
```

```
mov eax,x
```

```
call atoi
```

```
mov edx,eax
```

```
mov eax,vvoda
```

```
call sprint
```

```
mov ecx,a
```

```
mov edx,80
```

```
call sread
```

```
mov eax,a
```

```
call atoi
```

```
cmp eax,edx
```

```
jl _functionx
```

```
jmp _functiona
```

```
_functiona:
```

```
mov edx,3
```

```
mul edx
```

```
add eax,1
```

```
jmp _end
```

```
_functionx:
```

```
mov edx,3
```

```
mul edx
```

```
add eax,1
```

```
jmp _end
```

```
_end:
```

```
mov ecx,eax
```

```
mov eax,vivod
```

```
call sprint
```

```
mov eax,ecx
```

```
call iprintLF
```

```
call quit
```

## **5 Выводы**

В ходе данной лабораторной работы я изучила команды условного и безусловного переходов, научилась писать программы с использованием переходов, а также ознакомилась с назначением и структурой файла листинга.

# Список литературы

Архитектура ЭВМ