

Отчёт по лабораторной работе

Дисциплина: Архитектура компьютера

Карпова Анастасия Александровна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
5	Выводы	17
	Список литературы	18

Список иллюстраций

4.1	Создание файла	8
4.2	Ввод программы	8
4.3	Создание и запуск исполняемого файла	9
4.4	Изменение текста программы	10
4.5	Создание и запуск исполняемого файла	10
4.6	Изменение текста программы	11
4.7	Ввод программы	12
4.8	Создание и запуск исполняемого файла + проверка	12
4.9	Создание файла листинга	13
4.10	Открытие файла листинга	13
4.11	Содержимое	13
4.12	Открываю файл и удаляю операнд	13
4.13	ошибка	14
4.14	Работа программы	14
4.15	Работа программы	16

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. 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: • 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` (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. Как отмечалось выше, для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов. 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. Команда условного перехода имеет вид `j label` Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов. В табл. 7.3. представлены команды условного перехода, которые обычно ставятся после команды сравнения `cmp`. В их мнемокодах указывается тот результат сравнения, при котором надо делать пе-

реход. Мнемоники, идентичные по своему действию, написаны в таблице через дробь (например, `ja` и `jnbe`). Программист выбирает, какую из них применить, чтобы получить более простой для понимания текст программы. 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

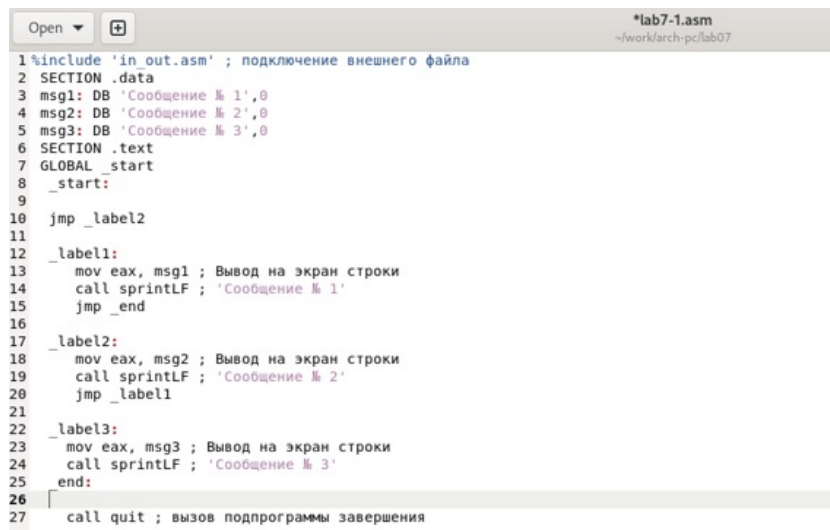
7.1. Реализация переходов в NASM

Создаю каталог для программ лабораторной работы № 7, перехожу в него и создаю файл lab7-1.asm: (рис. 4.1).

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

Рис. 4.1: Создание файла

Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`. Ввожу в файл lab7-1.asm текст программы из листинга 7.1. (рис. 4.2)



```
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
10 jmp _label2
11
12 _label1:
13     mov eax, msg1 ; Вывод на экран строки
14     call sprintf ; 'Сообщение № 1'
15     jmp _end
16
17 _label2:
18     mov eax, msg2 ; Вывод на экран строки
19     call sprintf ; 'Сообщение № 2'
20     jmp _label1
21
22 _label3:
23     mov eax, msg3 ; Вывод на экран строки
24     call sprintf ; 'Сообщение № 3'
25     end:
26
27     call quit ; вызов подпрограммы завершения
```

Рис. 4.2: Ввод программы

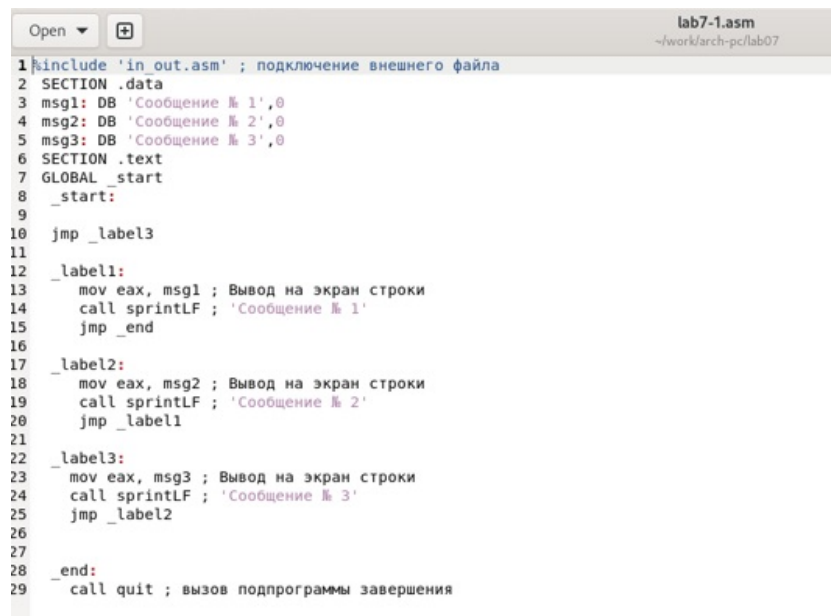
Создаю исполняемый файл и запускаю его. Результат работы данной программы будет следующим: Сообщение № 2 Сообщение № 3 (рис. 4.3)

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

Рис. 4.3: Создание и запуск исполняемого файла

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

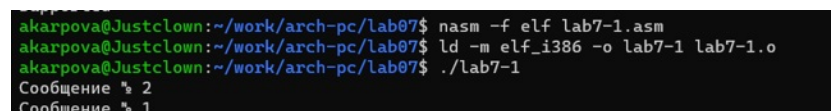
Инструкция `jmp` позволяет осуществлять переходы не только вперед но и назад. Изменим программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. Для этого в текст программы после вывода сообщения № 2 добавляю инструкцию `jmp` с меткой `_label1` (т.е. переход к инструкциям вывода сообщения № 1) и после вывода сообщения № 1 добавляю инструкцию `jmp` с меткой `_end` (т.е. переход к инструкции `call quit`). Изменяю текст программы в соответствии с листингом 7.2. (рис. 4.4)



```
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
10 jmp _label3
11
12 _label1:
13     mov eax, msg1 ; Вывод на экран строки
14     call sprintf ; 'Сообщение № 1'
15     jmp _end
16
17 _label2:
18     mov eax, msg2 ; Вывод на экран строки
19     call sprintf ; 'Сообщение № 2'
20     jmp _label1
21
22 _label3:
23     mov eax, msg3 ; Вывод на экран строки
24     call sprintf ; 'Сообщение № 3'
25     jmp _label2
26
27
28 _end:
29     call quit ; вызов подпрограммы завершения
```

Рис. 4.4: Изменение текста программы

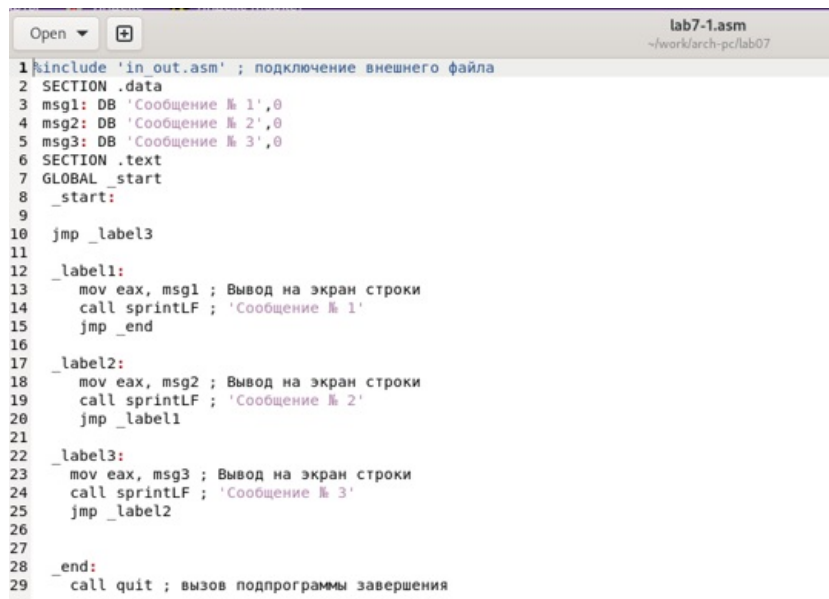
Создаю исполняемый файл и запускаю его. (рис. 4.5)



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

Рис. 4.5: Создание и запуск исполняемого файла

Изменяю текст программы добавив в начале программы `jmp_label3`(вместо `label2`), `jmp_label2` в конце метки `jmp_label3`, `jmp_label1` добавляю в конце метки `jmp_label2`, в конце метки `jmp_label1` добавляю `jmp_end`, чтобы вывод программы был следующим: Сообщение № 3 Сообщение № 2 Сообщение № 1 (рис. 4.6)



```

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
10 jmp _label3
11
12 _label1:
13     mov eax, msg1 ; Вывод на экран строки
14     call printf ; 'Сообщение № 1'
15     jmp _end
16
17 _label2:
18     mov eax, msg2 ; Вывод на экран строки
19     call printf ; 'Сообщение № 2'
20     jmp _label1
21
22 _label3:
23     mov eax, msg3 ; Вывод на экран строки
24     call printf ; 'Сообщение № 3'
25     jmp _label2
26
27
28 _end:
29     call quit ; вызов подпрограммы завершения

```

Рис. 4.6: Изменение текста 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.

Создайте файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. Внимательно изучаю текст программы из листинга 7.3 и ввожу в lab7-2.asm.

```

akarpova@Justclown:~/work/arch-pc/lab07$ touch lab7-2.asm
akarpova@Justclown:~/work/arch-pc/lab07$ gedit lab7-2.asm
*lab7-2.asm
~/work/arch-pc/lab07

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 iprintLF ; Вывод 'max(A,B,C)'
49     call quit

```

Рис. 4.7: Ввод программы

Создайте исполняемый файл и проверьте его работу для разных значений B.
(рис. 4.8)

```

akarpova@Justclown:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
akarpova@Justclown:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
akarpova@Justclown:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 12
Наибольшее число: 50
akarpova@Justclown:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 13
Наибольшее число: 50
akarpova@Justclown:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 123
Наибольшее число: 123
akarpova@Justclown:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 50
Наибольшее число: 50
akarpova@Justclown:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 51
Наибольшее число: 51

```

Рис. 4.8: Создание и запуск исполняемого файла + проверка

7.2. 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. Создаю файл листинга для программы из файла `lab7-2.asm`

```
akarpova@Justclown:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
```

Рис. 4.9: Создание файла листинга

Откройте файл листинга `lab7-2.lst` с помощью любого текстового редактора

```
akarpova@Justclown:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
```

Рис. 4.10: Открытие файла листинга

Внимательно ознакамливаюсь с его форматом и содержимым. В представленных трёх строках содержатся след. данные:

“2” - номер строки кода”; Функция вычисления длины 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. (рис. 4.11)

3	<1>	; Функция вычисления длины сообщения
4	<1>	slen:
5	00000000 53	push ebx

Рис. 4.11: Содержимое

Открываю файл с программой `lab7-2.asm` и в выбранной мной инструкции с двумя операндами удаляю выделенный операнд. (рис. 4.12).

```
mov [max],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'C' (как символ)
; ----- Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', то переход на метку
mov ecx, [C] ; иначе 'ecx = C'
```

Рис. 4.12: Открываю файл и удаляю операнд

Выполняю трансляцию с получением 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. (рис. 4.13)

```
akarpova@Justclown:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands
akarpova@Justclown:~/work/arch-pc/lab07$
```

Рис. 4.13: ошибка

7.3. Выполнение задания для самостоятельной работы

Пишу программу нахождения наименьшей из 3 целочисленных переменных а, b и с. Значения переменных выбираю из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Мой вариант под номером 15, поэтому мои значения - 32,6,54. (рис. 4.14)

```
akarpova@Justclown:~/work/arch-pc/lab07$ nasm -f elf lab7-3.asm
akarpova@Justclown:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-3.o -o lab7-3
akarpova@Justclown:~/work/arch-pc/lab07$ ./lab7-3
Наименьшее число: 6
```

Рис. 4.14: Работа программы

Код программы:

```
%include 'in_out.asm'

section .data
msg db "Наименьшее число:",0h
A dd '41'
B dd '62'
C dd '35'

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,[B] ; 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
mov [min],eax ; запись преобразованного числа в min
; ---- Сравниваем 'min(A,C)' и 'B' (как числа)
mov ecx,[min]
cmp ecx,[B] ; Сравниваем 'min(A,C)' и 'B'
jl fin ; если 'min(A,C)<B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
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 и выводит результат вычислений заданной для моего варианта функции $f(x)$:

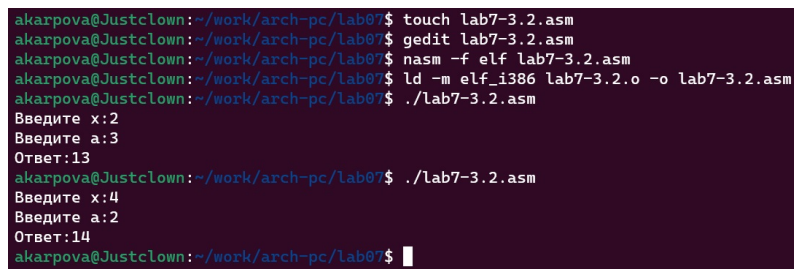
$a + 10, x < a \times 10, x \geq a$

%include 'in_out.asm' section .data msg1 db "Введите x:",0h msg2 db "Введите a:",0h msg3 db "Ответ:",0h section .bss x resb 10 a resb 10 section .text global _start
_start: mov eax,msg1 call sprint mov ecx,x mov edx,10 call sread mov eax,x call atoi ;
Вызов подпрограммы перевода символа в число mov [x],eax ; запись преобразованного числа в 'x'

mov eax,msg2 call sprint mov ecx,a mov edx,10 call sread ; ——— Преобразование
'a' из символа в число mov eax,a call atoi ; Вызов подпрограммы перевода символа
в число mov [a],eax ; запись преобразованного числа в 'a'

mov eax,[x] mov ebx,[a] cmp eax,ebx jl fin jmp fin1 fin: mov eax, msg3 call sprint
mov eax,[a] add eax, 10 call iprintLF call quit ; Выход fin1: mov eax,msg3 call sprint
mov eax,[x] add eax, 10 call iprintLF call quit ; Выход

Запускаю файл (рис. 4.15)



```
akarpova@Justclown: ~/work/arch-pc/lab0$ touch lab7-3.2.asm
akarpova@Justclown: ~/work/arch-pc/lab0$ gedit lab7-3.2.asm
akarpova@Justclown: ~/work/arch-pc/lab0$ nasm -f elf lab7-3.2.asm
akarpova@Justclown: ~/work/arch-pc/lab0$ ld -m elf_i386 lab7-3.2.o -o lab7-3.2.asm
akarpova@Justclown: ~/work/arch-pc/lab0$ ./lab7-3.2.asm
Введите x:2
Введите a:3
Ответ:13
akarpova@Justclown: ~/work/arch-pc/lab0$ ./lab7-3.2.asm
Введите x:4
Введите a:2
Ответ:14
akarpova@Justclown: ~/work/arch-pc/lab0$
```

Рис. 4.15: Работа программы

5 Выводы

В ходе данной лабораторной работы я изучила команды условного и безусловного переходов, приобрела навыки написания программ с использованием переходов и ознакомилась с назначением и структурой файла листинга, что поможет мне при выполнении последующих лабораторных работ.

Список литературы