

Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютера

Кайнова Алина Андреевна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
4.1	Реализация переходов в NASM	8
4.2	Изучение структуры файла листинга	13
4.3	Выполнение заданий для самостоятельной работы	15
5	Выводы	20
	Список литературы	21

Список иллюстраций

4.1	Создание каталога	8
4.2	Заполнение файла	9
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	Создание и запуск исполняемого файла	11
4.8	Создание файла	12
4.9	Ввод текста программы	12
4.10	Создание и запуск исполняемого файла	13
4.11	Создание и открытие файла	13
4.12	Изучение файла листинга	14
4.13	Выбранные строки файла	14
4.14	Удаление выделенного операнда из кода	15
4.15	Получение файла листинга	15
4.16	Листинг после удаления	15
4.17	Написание программы	16
4.18	Создание файла и проверка его работы	17
4.19	Написание программы	18
4.20	Создание файла и проверка его работы	19

1 Цель работы

Изучить команды переходов в ассемблере, приобрести навыки написания программ с обработкой аргументов командной строки и познакомиться с файлом листинга.

2 Задание

1. Реализация циклов в NASM
2. Изучение структуры файла 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. Адрес перехода – это либо метка, либо адрес области памяти. в качестве операнда можно использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре. 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` позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Команда условного перехода имеет вид `j label` Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов. Команды условного перехода обычно ставятся после команды сравнения `cmp`. В их мнемокодах указывается тот результат сравнения, при котором надо

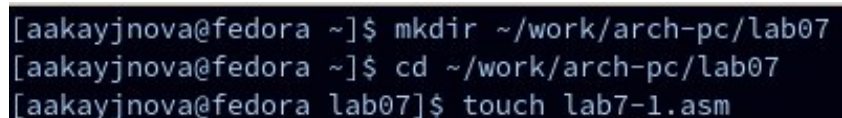
делать переход. Программист выбирает, какую из мнемоник применить, чтобы получить более простой для понимания текст программы. Листинг (в рамках понятийного аппарата NASM) — это один из выходных файлов, создаваемых транслятором. Он имеет текстовый вид и нужен при отладке программы, так как кроме строк самой программы он содержит дополнительную информацию: сообщения об ошибках и предупреждения, обнаруженные при ассемблировании, транслятор выводит на экран, и файл листинга не создаётся. Структура листинга:

- номер строки — это номер строки файла листинга (нужно помнить, что номер строки в файле листинга может не соответствовать номеру строки в файле с исходным текстом программы);
- адрес — это смещение машинного кода от начала текущего сегмента;
- машинный код представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности. (например, инструкция `int 80h` начинается по смещению `00000020` в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция `int 80h` ассемблируется в `CD80` (в шестнадцатеричном представлении); `CD80` — это инструкция на машинном языке, вызывающая прерывание ядра);
- исходный текст программы — это просто строка исходной программы вместе с комментариями (некоторые строки на языке ассемблера, например, строки, содержащие только комментарии, не генерируют никакого машинного кода, и поля «смещение» и «исходный текст программы» в таких строках отсутствуют, однако номер строки им присваивается).

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

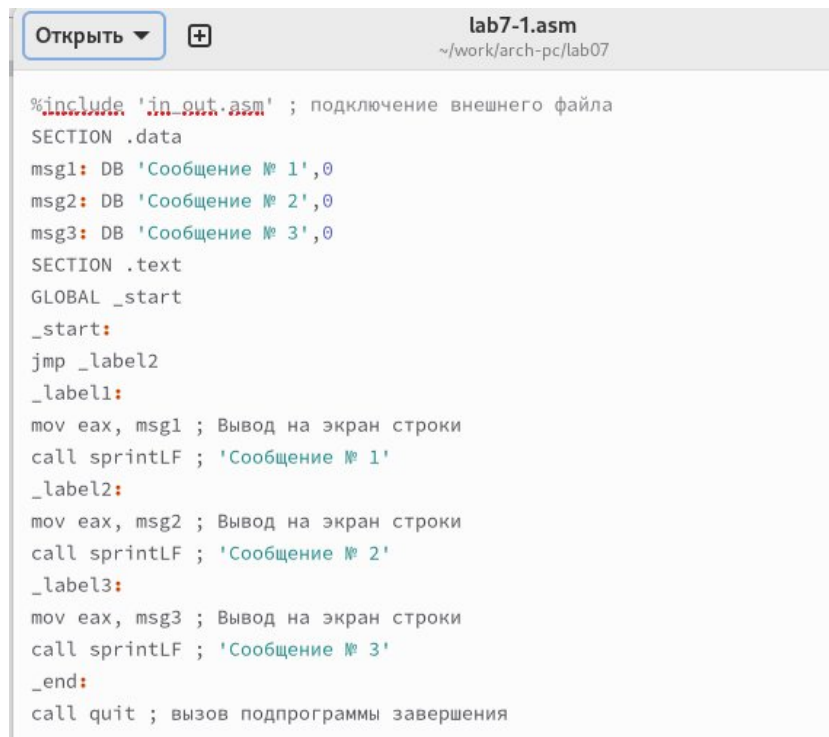
Создаю каталог для лабораторной работы №7 и в нём файл lab7-1.asm



```
[aakayjnova@fedora ~]$ mkdir ~/work/arch-pc/lab07  
[aakayjnova@fedora ~]$ cd ~/work/arch-pc/lab07  
[aakayjnova@fedora lab07]$ touch lab7-1.asm
```

Рис. 4.1: Создание каталога

Заполняю файл, вставляя в него текст программы из листинга 7.1

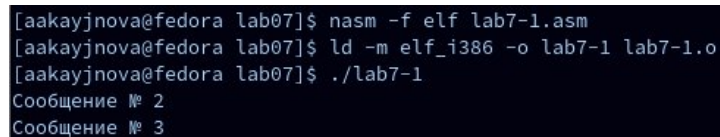


```
Открыть ▾  lab7-1.asm
~/work/arch-pc/lab07

%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 sprintf ; 'Сообщение № 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
```

Рис. 4.2: Заполнение файла

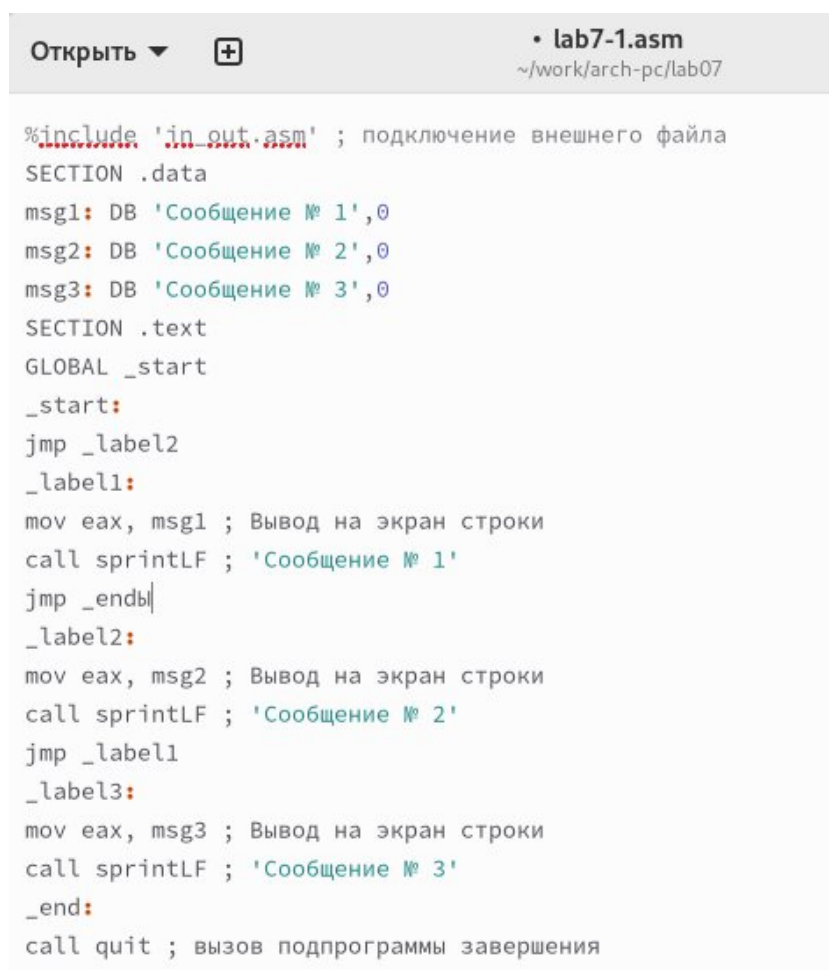
Создаю исполняемый файл и запускаю его



```
[aakayjnova@fedora lab07]$ nasm -f elf lab7-1.asm
[aakayjnova@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[aakayjnova@fedora lab07]$ ./lab7-1
Сообщение № 2
Сообщение № 3
```

Рис. 4.3: Создание и запуск исполняемого файла

Изменяю текст программы в соответствии с листингом 7.2

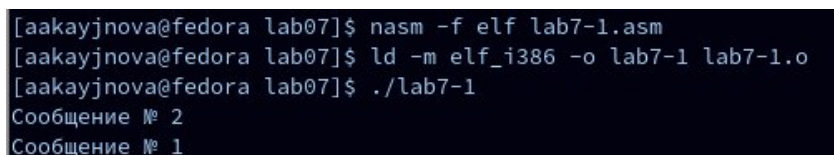


```
Открыть ▾ + • lab7-1.asm
~/work/arch-pc/lab07

%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 printf ; 'Сообщение № 1'
jmp _endb1
_label2:
mov eax, msg2 ; Вывод на экран строки
call printf ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call printf ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
```

Рис. 4.4: Изменение текста программы

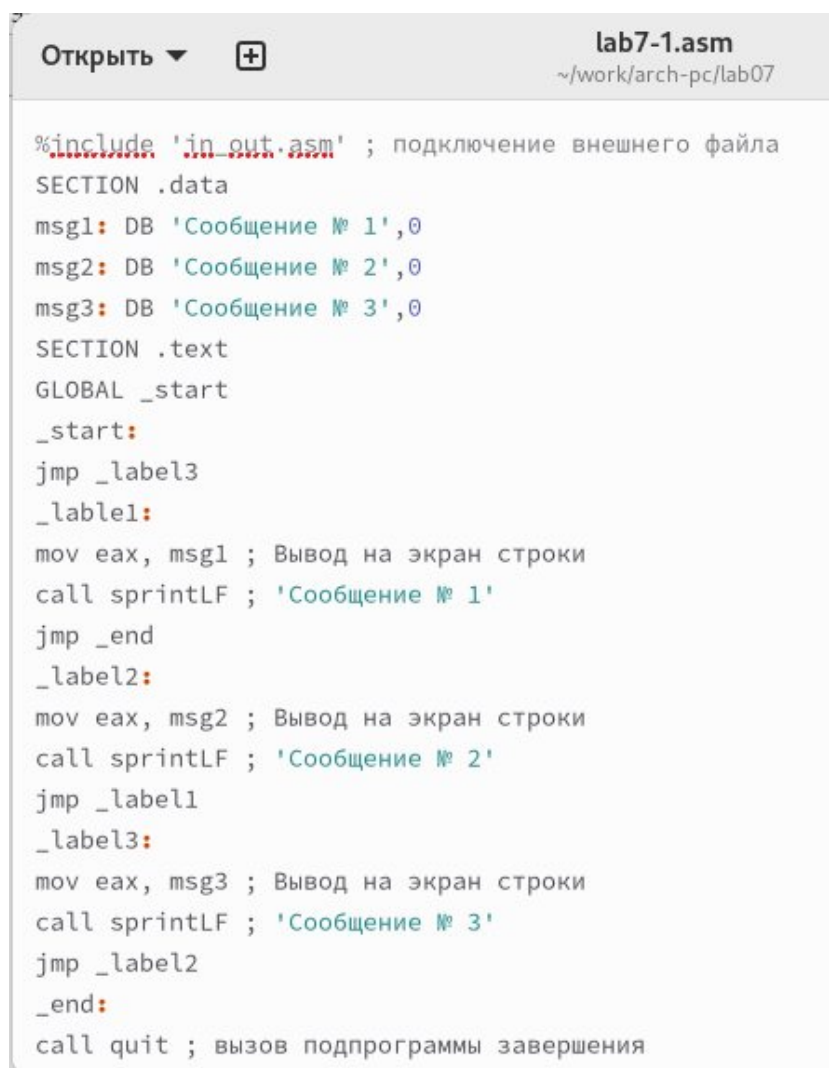
Создаю исполняемый файл и проверяю его работу



```
[aakayjnova@fedora lab07]$ nasm -f elf lab7-1.asm
[aakayjnova@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[aakayjnova@fedora lab07]$ ./lab7-1
Сообщение № 2
Сообщение № 1
```

Рис. 4.5: Создание и запуск исполняемого файла

Изменяю текст программы так, чтобы вывод программы соответствовал заданию




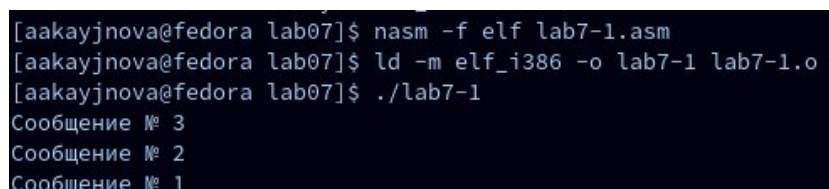
```
Открыть ▾  lab7-1.asm  
~/work/arch-pc/lab07  
  
%include 'in_out.asm' ; подключение внешнего файла  
SECTION .data  
msg1: DB 'Сообщение № 1',0  
msg2: DB 'Сообщение № 2',0  
msg3: DB 'Сообщение № 3',0  
SECTION .text  
GLOBAL _start  
_start:  
jmp _label3  
_label1:  
mov eax, msg1 ; Вывод на 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'  
jmp _label2  
_end:  
call quit ; вызов подпрограммы завершения
```

Рис. 4.6: Изменение текста программы

Создаю исполняемый файл и проверяю его работу



```
[aakayjnova@fedora lab07]$ nasm -f elf lab7-1.asm  
[aakayjnova@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o  
[aakayjnova@fedora lab07]$ ./lab7-1  
Сообщение № 3  
Сообщение № 2  
Сообщение № 1
```

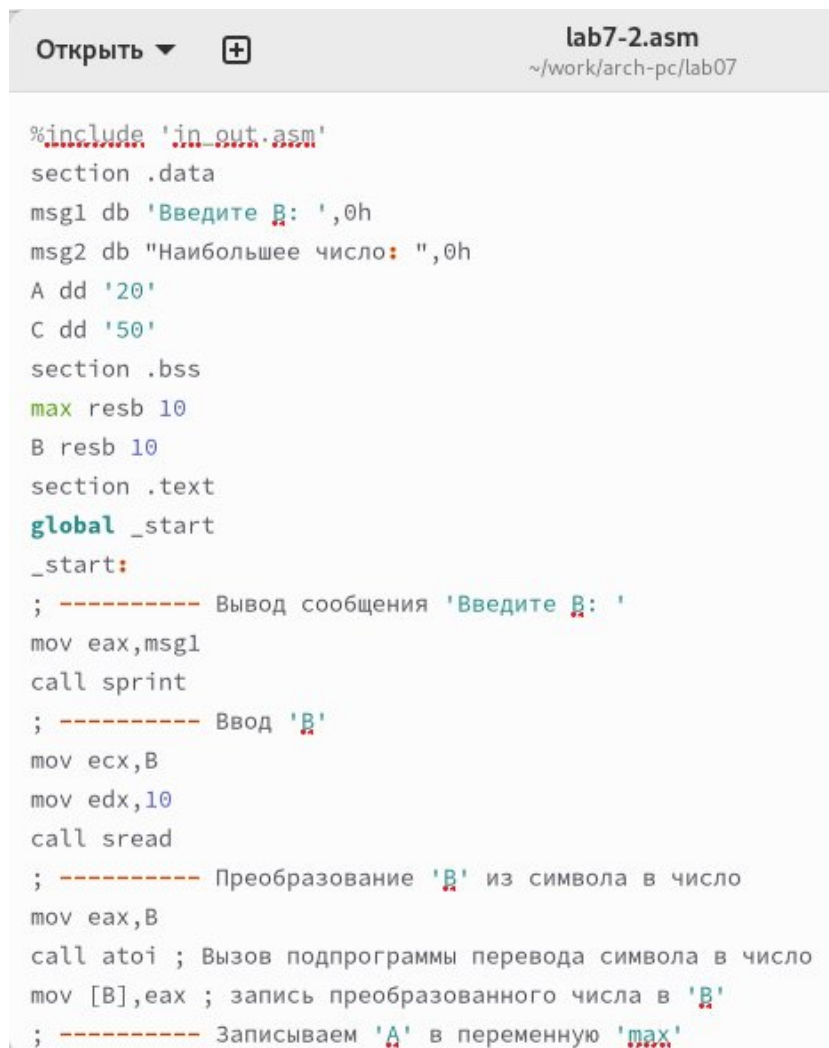
Рис. 4.7: Создание и запуск исполняемого файла

Создаю файл lab7-2.asm

```
[aakayjnova@fedora lab07]$ touch lab7-2.asm
[aakayjnova@fedora lab07]$ ls
in_out.asm  lab7-1  lab7-1.asm  lab7-1.o  lab7-2.asm
```

Рис. 4.8: Создание файла

Ввожу в него текст программы из листинга 7.3



```
lab7-2.asm
~/work/arch-pc/lab07

%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'
```

Рис. 4.9: Ввод текста программы

Создаю исполняемый файл и проверяю его работу

```
[aakayjnova@fedora lab07]$ nasm -f elf lab7-2.asm
[aakayjnova@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[aakayjnova@fedora lab07]$ ./lab7-2
Введите В: 100
Наибольшее число: 100
[aakayjnova@fedora lab07]$ ./lab7-2
Введите В: 5
Наибольшее число: 50
```

Рис. 4.10: Создание и запуск исполняемого файла

4.2 Изучение структуры файла листинга

Создаю файл листинга для программы из файла lab7-2.asm и открою созданный файл

```
[aakayjnova@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
[aakayjnova@fedora lab07]$ gedit lab7-2.lst
```

Рис. 4.11: Создание и открытие файла

Изучаю формат и содержимое данного файла листинга

Line	Address	Hex	Comment / Instruction
1	1		%include 'in_out.asm'
2	1		<1> ;----- slen -----
3	2		<1> ; Функция вычисления длины сообщения
4	3		<1> slen:
5	4	00000000 53	<1> push ebx
6	5	00000001 89C3	<1> mov ebx, eax
7	6		<1>
8	7		<1> nextchar:
9	8	00000003 803800	<1> cmp byte [eax], 0
10	9	00000006 7403	<1> jz finished
11	10	00000008 40	<1> inc eax
12	11	00000009 EBF8	<1> jmp nextchar
13	12		<1>
14	13		<1> finished:
15	14	00000008 29D8	<1> sub eax, ebx
16	15	0000000D 5B	<1> pop ebx
17	16	0000000E C3	<1> ret
18	17		<1>
19	18		<1>
20	19		<1> ;----- sprint -----
21	20		<1> ; Функция печати сообщения
22	21		<1> ; входные данные: mov eax,<message>
23	22		<1> sprint:
24	23	0000000F 52	<1> push edx
25	24	00000010 51	<1> push ecx
26	25	00000011 53	<1> push ebx
27	26	00000012 50	<1> push eax
28	27	00000013 E8E8FFFFFF	<1> call slen
29	28		<1>
30	29	00000018 89C2	<1> mov edx, eax
31	30	0000001A 58	<1> pop eax
32	31		<1>
33	32	00000018 89C1	<1> mov ecx, eax
34	33	0000001D B801000000	<1> mov ebx, 1
35	34	00000022 B804000000	<1> mov eax, 4
36	35	00000027 CD80	<1> int 80h

Рис. 4.12: Изучение файла листинга

Представленные три строки содержат следующие данные

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

Рис. 4.13: Выбранные строки файла

строка № 2: “;Функция вычисления длины сообщения” - это комментарий к коду, нет адреса и машинного кода; строка № 3: “slen” - название функции, нет адреса и машинного кода; строка № 4: “00000000” - адрес строки, “53” - машинный код, “push ebx” - исходный текст программы(push помещает ebx в стек);

Открываю файл листинга и в любой инструкции с двумя операндами удаляю один операнд

```

27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 0000011C 3B0D[39000000] cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 00000122 7F0C jg check_B ; если 'A>C', то переход на метку 'check_B',

```

Рис. 4.14: Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга

```

[aakayjnova@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
[aakayjnova@fedora lab07]$ ls
in_out.asm lab7-1 lab7-1.asm lab7-1.o lab7-2 lab7-2.asm lab7-2.lst lab7-2.o

```

Рис. 4.15: Получение файла листинга

В этом случае создаётся файл листинга lab7-2.lst и в листинг добавляется операнд, который был удалён ранее

```

27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 0000011C 3B0D[39000000] cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 00000122 7F0C jg check_B ; если 'A>C', то переход на метку 'check_B',

```

Рис. 4.16: Листинг после удаления

4.3 Выполнение заданий для самостоятельной работы

Пишу программу нахождения наименьшей из 3-х целочисленных переменных a, b и c. Значения переменных выбираю из таблицы 7.5 в соответствии с вариантом, полученным в ходе выполнения лабораторной работы № 6. Мой вариант № 5, поэтому мои значения: 54, 62, 87.


```

mov eax,msgC call sprint mov ecx,C mov edx,80 call sread mov eax,C call atoi mov
[C],eax
mov ecx,[A] mov [min],ecx
cmp ecx, [B] jl check_C mov ecx, [B] mov [min], ecx
check_C: cmp ecx, [C] jl finish mov ecx,[C] mov [min],ecx
finish: mov eax,msgSM call sprint
mov eax, [min] call iprintLF
call quit

```

Создаю исполняемый файл и проверяю его работу

```

[aakayjnova@fedora lab07]$ nasm -f elf task1.asm
[aakayjnova@fedora lab07]$ ld -m elf_i386 task1.o -o task1
[aakayjnova@fedora lab07]$ ./task1
Введите значение A: 54
Введите значение B: 62
Введите значение C: 87
Ответ: 54

```

Рис. 4.18: Создание файла и проверка его работы

Код программы работает корректно.

Пишу программу, которая для введённых с клавиатуры значений x и a вычисляет значение заданной функции $f(x)$ и выводит результат вычислений. Мой вариант № 5, поэтому моя функция: $2(x-a)$, $x > a$; 15 , $x \leq a$;

```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msgX: DB 'Введите x: ',0
5 msgA: DB 'Введите а: ',0
6 answer: DB 'Результат: ',0
7
8 SECTION .bss
9 X: RESB 80
10 A: RESB 80
11 result: RESB 80
12
13 SECTION .text
14 GLOBAL _start
15
16 _start:
17 mov eax,msgX
18 call sprint
19 mov ecx,X
20 mov edx,80
21 call sread
22 mov eax,X
23 call atoi
24 mov [X],eax
25
26 mov eax,msgA
27 call sprint
28 mov ecx,A
```

Рис. 4.19: Написание программы

Текст программы: %include 'in_out.asm'

SECTION .data msgX: DB 'Введите x:',0 msgA: DB 'Введите а:',0 answer: DB 'Результат:',0

SECTION .bss X: RESB 80 A: RESB 80 result: RESB 80

SECTION .text GLOBAL _start

_start: mov eax,msgX call sprint mov ecx,X mov edx,80 call sread mov eax,X call
atoi mov [X],eax

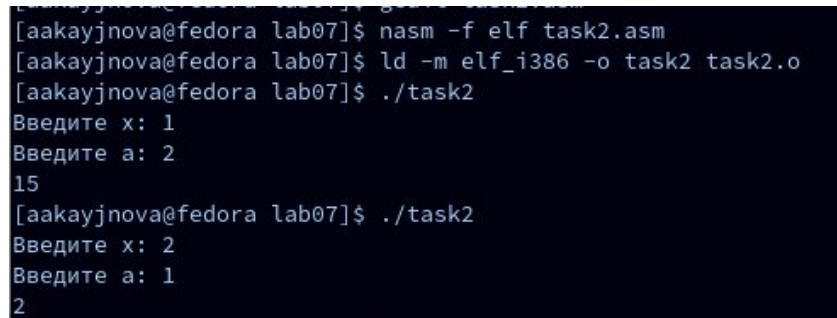
mov eax,msgA call sprint mov ecx,A mov edx,80 call sread mov eax,A call atoi mov
[A],eax

mov ecx,[X] cmp ecx,[A] JG first jmp secon first: mov eax,[X] sub eax,[A] mov ebx,2

mul ebx call iprintLF call quit

second: mov eax,15 call iprintLF call quit

Создаю исполняемый файл и проверяю его работу для значений x и a соответственно: (1;2) и (2;1)



```
[aakayjnova@fedora lab07]$ nasm -f elf task2.asm
[aakayjnova@fedora lab07]$ ld -m elf_i386 -o task2 task2.o
[aakayjnova@fedora lab07]$ ./task2
Введите x: 1
Введите a: 2
15
[aakayjnova@fedora lab07]$ ./task2
Введите x: 2
Введите a: 1
2
```

Рис. 4.20: Создание файла и проверка его работы

Код программы работает корректно.

5 Выводы

В ходе данной лабораторной работы мы изучили команды условного и безусловного переходов, приобрели навыки написания программ с их использованием и ознакомились со структурой файла листинга.

Список литературы

- [illegible]