

Шаблон отчёта по лабораторной работе

Простейший вариант

Дмитрий Сергеевич Кулябов

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Реализация переходов в NASM	9
4.2	Изучение структуры файлы листинга	13
4.3	Выполнение задания для самостоятельной работы	15
4.3.1	Задание 1	15
4.3.2	Задание 2	17
5	Выводы	19
6	Список литературы	20

Список иллюстраций

4.1	Создание каталога lab07 и файла lab7-1.asm в нем	9
4.2	Копирование файла in_out.asm	9
4.3	Проверка, что файл находится в нужном каталоге	10
4.4	Ввод текста программы в файл	10
4.5	Создание исполняемого файла и его запуск	10
4.6	Изменение текста программы	11
4.7	Создание исполняемого файла и его запуск	11
4.8	Изменение текста программы	12
4.9	Создание исполняемого файла и его запуск	12
4.10	Создание файла lab7-2.asm	12
4.11	Ввод текста программы в файл	13
4.12	Создание исполняемого файла и его запуск	13
4.13	Создание файла листинга	14
4.14	Открытие файла листинга	14
4.15	Строки из файла листинга	14
4.16	Инструкция mov с двумя операндами	15
4.17	Инструкция mov с одним операндом	15
4.18	Трансляция файла	15
4.19	Файл листинга	15
4.20	Создание файла var7-1.asm	16
4.21	Ввод текста программы	16
4.22	Ввод текста программы	17
4.23	Создание исполняемого файла и его запуск	17
4.24	Создание файла var7-2.asm	17
4.25	Ввод текста программы	18
4.26	Проверка работа программы для значений $x = 2$, $a = 1$	18

Список таблиц

1 Цель работы

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

2 Задание

1. Реализация переходов в NASM
2. Изучение структуры файлы листинга
3. Задание 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>`

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.

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 <операнд_1>, <операнд_2>` Единственным результатом команды сравнения является формирование флагов.

Команда условного перехода имеет вид `j<мнемоника_перехода> label` Мне-

моника 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

Все ошибки и предупреждения, обнаруженные при ассемблировании, транслятор выводит на экран, и файл листинга не создаётся. Итак, структура листинга:

- номер строки — это номер строки файла листинга;
- адрес — это смещение машинного кода от начала текущего сегмента;
- машинный код представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности.;
- исходный текст программы — это просто строка исходной программы вместе с комментариями.

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

Создаю каталог lab07 для программ лабораторной работы №7, перехожу в него и создаю файл lab7-1.asm, проверяю, что файл создан (рис. 4.1)

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

Рис. 4.1: Создание каталога lab07 и файла lab7-1.asm в нем

Перед работой с программами копирую файл in_out.asm в каталог и проверяю, что файл находится в нужном каталоге (рис. 4.2) (рис. 4.3)

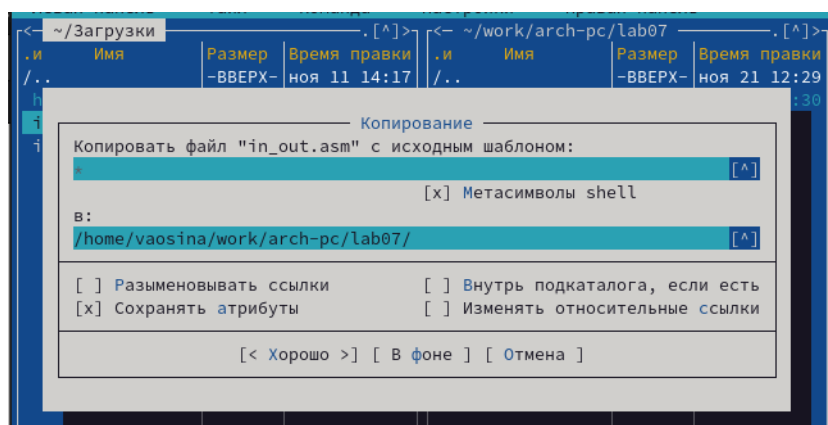


Рис. 4.2: Копирование файла in_out.asm

```
[vaosina@fedora lab07]$ ls
in_out.asm  lab7-1.asm
[vaosina@fedora lab07]$
```

Рис. 4.3: Проверка, что файл находится в нужном каталоге

Ввожу в файл lab7-1.asm текст программы (рис. 4.4).



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

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

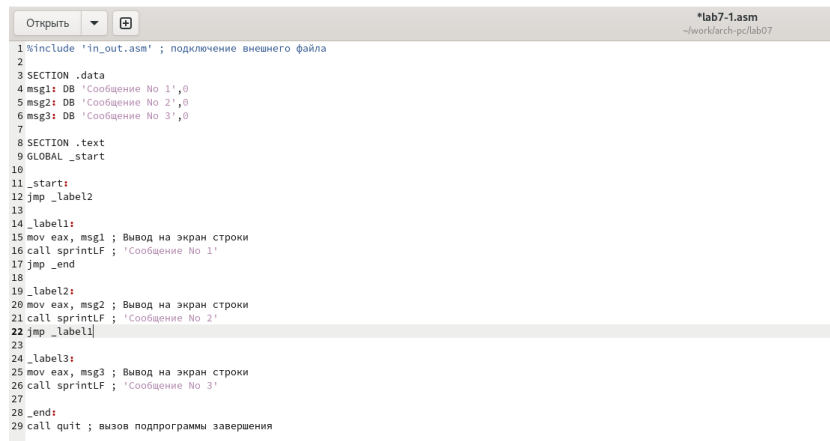
Рис. 4.4: Ввод текста программы в файл

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

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

Рис. 4.5: Создание исполняемого файла и его запуск

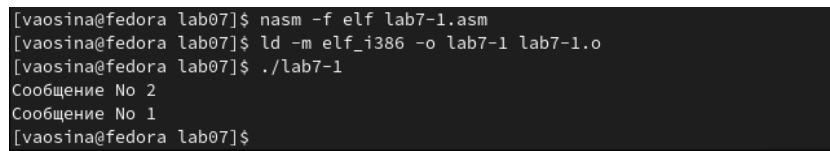
Изменяю текст программы таким образом, чтобы она выводила сначала ‘Сообщение No 2’, потом ‘Сообщение No 1’ и завершала работу. (рис. 4.6).



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

Рис. 4.6: Изменение текста программы

Создаю исполняемый файл и запускаю его. (рис. 4.7) Действительно, сначала выводится ‘Сообщение No 2’, потом ‘Сообщение No 1’



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

Рис. 4.7: Создание исполняемого файла и его запуск

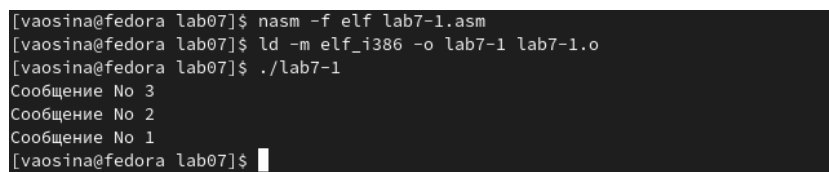
Теперь изменяю текст программы таким образом, чтобы она выводила сначала ‘Сообщение No 3’, потом ‘Сообщение No 2’, за ним ‘Сообщение No 1’ и завершала работу. (рис. 4.8).



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

Рис. 4.8: Изменение текста программы

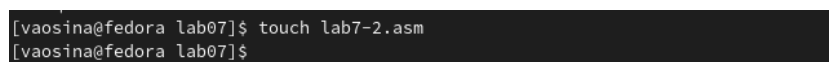
Создаю исполняемый файл и запускаю его. (рис. 4.9). Действительно, сначала выводится ‘Сообщение No 3’, потом ‘Сообщение No 2’, потом ‘Сообщение No 1’



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

Рис. 4.9: Создание исполняемого файла и его запуск

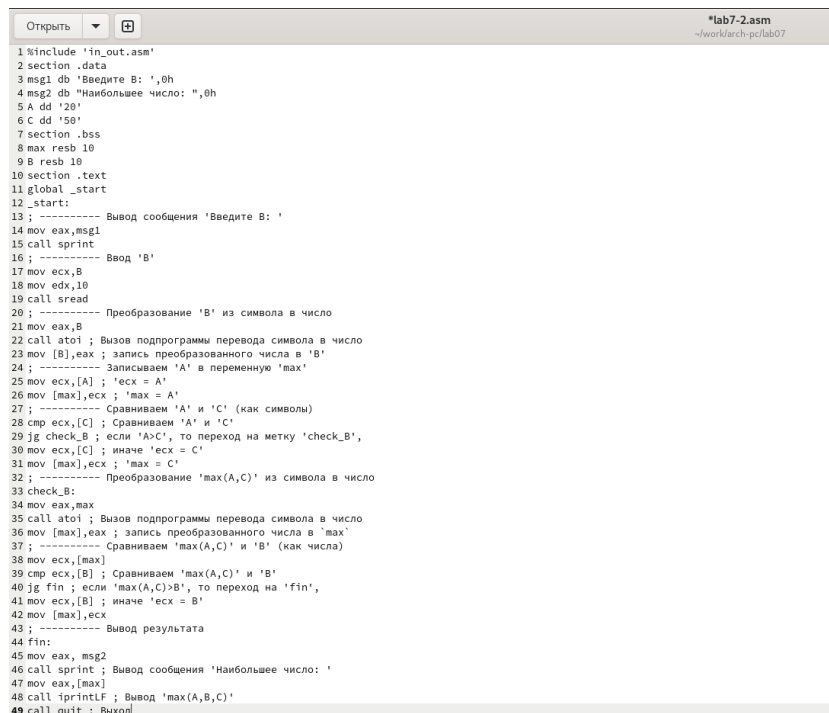
Создаю файл lab7-2.asm в каталоге ~/work/arch-pc/lab07 (рис. 4.10)



```
[vaosina@fedora lab07]$ touch lab7-2.asm
[vaosina@fedora lab07]$
```

Рис. 4.10: Создание файла lab7-2.asm

Ввожу в файл текст программы

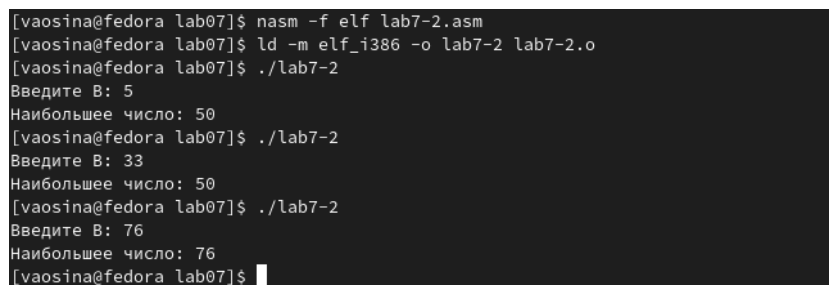


```
Открыть [иконка] *lab7-2.asm
~work/arch-pc/lab07

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,8
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 'printf' ; Вывод 'max(A,B,C)'
49 call quit ; Выход
```

Рис. 4.11: Ввод текста программы в файл

Создаю исполняемый файл и запускаю его.(рис. 4.12). Проверяю его работу для разных значений B.



```
[vaosina@fedora lab07]$ nasm -f elf lab7-2.asm
[vaosina@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[vaosina@fedora lab07]$ ./lab7-2
Введите B: 5
Наибольшее число: 50
[vaosina@fedora lab07]$ ./lab7-2
Введите B: 33
Наибольшее число: 50
[vaosina@fedora lab07]$ ./lab7-2
Введите B: 76
Наибольшее число: 76
[vaosina@fedora lab07]$
```

Рис. 4.12: Создание исполняемого файла и его запуск

4.2 Изучение структуры файлы листинга

Создаю файл листинга для программы из файла lab7-2.asm.(рис. 4.13).

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

Рис. 4.13: Создание файла листинга

Открываю файл листинга lab7-2.lst (рис. 4.14).

```

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 EBFA                      <1> jmp     nextchar
13 12                                     <1>
14 13                                     <1> finished:
15 14 00000008 2908                     <1> sub     eax, ebx
16 15 00000000 58                       <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 EBFAFFFF                 <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 0000001B 89C1                     <1> mov     ecx, eax
34 33 0000001D 8B01000000               <1> mov     ebx, 1
35 34 00000022 8B04000000               <1> mov     eax, 4
36 35 00000027 CD80                     <1> int     80h
37 36                                     <1>
38 37 00000029 58                       <1> pop     ebx
39 38 0000002A 59                       <1> pop     ecx
40 39 0000002B 5A                       <1> pop     edx
41 40 0000002C C3                       <1> ret
42 41                                     <1>
43 42                                     <1>
44 43                                     <1> ;----- sprintLF -----
45 44                                     <1> ; функция печати сообщения с переводом строки
46 45                                     <1> ; входные данные: mov eax, <message>
47 46                                     <1> sprintLF:
48 47 0000002D EBFAFFFF                 <1> call    sprint
49 48                                     <1>
50 49 00000032 50                       <1> push    eax

```

Рис. 4.14: Открытие файла листинга

Привожу три строки из файла листинга для их описания (рис. 4.15) 1. 87 - номер строки; 00000068 - адрес строки; F7FE - машинный код; idiv esi - исходный текст программы, где idiv является инструкцией для знакового деления. 2. 88 - номер строки; 0000006A - адрес строки; 83C230 - машинный код; add edx, 48 - исходный текст программы, где add является инструкцией для сложения. 3. 89 - номер строки; 0000006D - адрес строки; 52 - машинный код; push edx - исходный текст программы, где push является инструкцией для помещения операнда в стек.

88	87 00000068 F7FE	<1>	idiv	esi
89	88 0000006A 83C230	<1>	add	edx, 48
90	89 0000006D 52	<1>	push	edx

Рис. 4.15: Строки из файла листинга

Открываю файл с программой lab7-2.asm и в инструкции mov с двумя операндами удаляю один операнд. (рис. 4.16) и (рис. 4.17)

```
14 mov eax,msg1
```

Рис. 4.16: Инструкция mov с двумя операндами

```
14 mov eax
```

Рис. 4.17: Инструкция mov с одним операндом

Выполняю трансляцию с получением файла листинга.(рис. 4.18) Выдается ошибка из-за нарушения работы кода, т. к. инструкция mov может работать только при наличии двух операндов

```
[vaosina@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:14: error: invalid combination of opcode and operands
[vaosina@fedora lab07]$
```

Рис. 4.18: Трансляция файла

В файле листинга это выглядит следующим образом (рис. 4.19)

```
189 14 mov eax
190 14 ***** error: invalid combination of opcode and operands
191 15 000000E8 E822FFFFFF call sprint
```

Рис. 4.19: Файл листинга

4.3 Выполнение задания для самостоятельной работы

4.3.1 Задание 1

Необходимо написать программу нахождения наименьшей из 3 целочисленных переменных a, b и c В соответствии с вариантом (7), полученным при выпол-

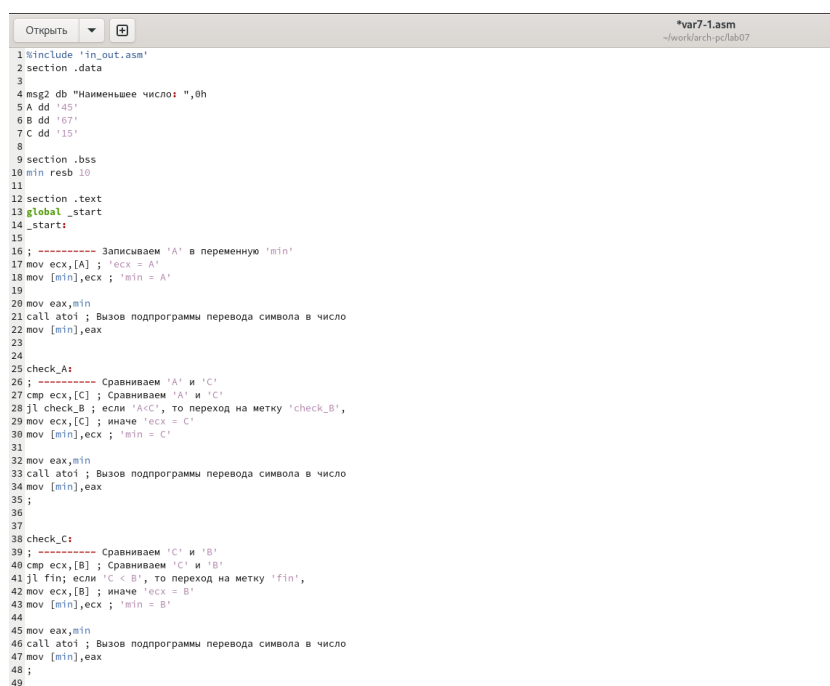
нении предыдущей лабораторной работы, значения переменных таковы: a = 45, b = 67, c = 15

Создаю файл var7-1.asm и открываю его (рис. 4.20).

```
[vaosina@fedora lab07]$ cp lab7-2.asm var7-1.asm
[vaosina@fedora lab07]$ ls
in_out.asm  lab7-1.asm  lab7-2      lab7-2.lst  var7-1.asm
lab7-1      lab7-1.o    lab7-2.asm  lab7-2.o
[vaosina@fedora lab07]$ gedit var7-1.asm
```

Рис. 4.20: Создание файла var7-1.asm

Ввожу текст программы (рис. 4.21) (рис. 4.22)



```
1 %include 'in_out.asm'
2 section .data
3
4 msg2 db "Наименьшее число: ",0h
5 A dd '45'
6 B dd '67'
7 C dd '15'
8
9 section .bss
10 min resb 10
11
12 section .text
13 global _start
14 _start:
15
16 ; ----- Записываем 'A' в переменную 'min'
17 mov ecx,[A] ; 'ecx = A'
18 mov [min],ecx ; 'min = A'
19
20 mov eax,min
21 call atoi ; Вызов подпрограммы перевода символа в число
22 mov [min],eax
23
24
25 check_A:
26 ; ----- Сравниваем 'A' и 'C'
27 cmp ecx,[C] ; Сравниваем 'A' и 'C'
28 jl check_B ; если 'A < C', то переход на метку 'check_B',
29 mov ecx,[C] ; иначе 'ecx = C'
30 mov [min],ecx ; 'min = C'
31
32 mov eax,min
33 call atoi ; Вызов подпрограммы перевода символа в число
34 mov [min],eax
35 ;
36
37
38 check_C:
39 ; ----- Сравниваем 'C' и 'B'
40 cmp ecx,[B] ; Сравниваем 'C' и 'B'
41 jl fin ; если 'C < B', то переход на метку 'fin',
42 mov ecx,[B] ; иначе 'ecx = B'
43 mov [min],ecx ; 'min = B'
44
45 mov eax,min
46 call atoi ; Вызов подпрограммы перевода символа в число
47 mov [min],eax
48 ;
49
```

Рис. 4.21: Ввод текста программы


```

50
51 check_B:
52 ; ----- Сравниваем 'A' и 'B'
53 cmp ecx,[B] ; Сравниваем 'A' и 'C'
54 jl fin; если 'A<B', то переход на метку 'fin',
55 mov ecx,[B] ; иначе 'ecx = B'
56 mov [min],ecx ; 'min = B'
57
58 mov eax,min
59 call atoi ; Вызов подпрограммы перевода символа в число
60 mov [min],eax
61 ;
62
63
64 ; ----- Вывод результата
65 fin:
66 mov eax, msg2
67 call sprint ; Вывод сообщения 'Наименьшее число: '
68 mov eax,[min]
69 call iprintLF ; Вывод 'min(A,B,C)'
70 call quit ; Выход

```

Рис. 4.22: Ввод текста программы

Создаю исполняемый файл и запускаю его. (рис. 4.23). Выводится значение 15. Программа работает корректно.

```

[vaosina@fedora lab07]$ nasm -f elf var7-1.asm
[vaosina@fedora lab07]$ ld -m elf_i386 -o var7-1 var7-1.o
[vaosina@fedora lab07]$ ./var7-1
Наименьшее число: 15
[vaosina@fedora lab07]$

```

Рис. 4.23: Создание исполняемого файла и его запуск

4.3.2 Задание 2

Необходимо написать программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции $f(x)$ и выводит результат вычислений. Функция: $6a$, если $x=a$; $a + x$, если $x \neq a$

Создаю файл var7-2.asm (рис. 4.24).

```

[vaosina@fedora lab07]$ cp lab7-2.asm var7-2.asm
[vaosina@fedora lab07]$ ls
in_out.asm  lab7-1.asm  lab7-2      lab7-2.lst  var7-1      var7-1.o
lab7-1      lab7-1.o    lab7-2.asm  lab7-2.o    var7-1.asm  var7-2.asm
[vaosina@fedora lab07]$

```

Рис. 4.24: Создание файла var7-2.asm

Ввожу в файл текст программы (рис. 4.25).

```

Открыть  ▾  [icon]  *var7-2.asm  ~/work/arch-pclab07
1 #include "in_out.asm"
2 section .data
3 msg1 db 'Введите x: ',0h
4 msg2 db 'Введите a: ',0h
5 msg3 db 'Результат: ',0h
6 section .bss
7 x: resb 80
8 a: resb 80
9 section .text
10 global _start
11 _start:
12
13 mov eax,msg1
14 call sprint
15
16 mov ecx,x
17 mov edx,80
18 call sread
19
20 mov eax,x
21 call atoi
22 mov [x], eax
23
24 mov eax, msg2
25 call sprint
26
27 mov ecx,a
28 mov edx,80
29 call sread
30
31 mov eax,a
32 call atoi
33 mov [a], eax
34
35 cmp eax,[x]
36 je _fx
37 add eax,[x]
38 jmp _fin
39
40 _fx:
41 mov edx,6
42 mul edx
43
44 _fin:
45 mov ecx, eax
46 mov eax, msg3
47 call sprint ; Вывод сообщения 'Результат: '
48 mov eax,ecx
49 call iprintf
50 call quit

```

Рис. 4.25: Ввод текста программы

Создаю исполняемый файл и запускаю его. (рис. ??). Проверяю работу программы для значений $x = 1$ и $a = 1$. Результат корректный ($x = a \Rightarrow 6a = 6 * 1 = 6$)

[Создание исполняемого файла и его запуск. Проверка работа программы для значений $x = 1, a = 1$] (image/26.png){#fig:026 width=70%}

Теперь проверяю работу программы для значений $x = 2$ и $a = 1$. (рис. 4.26). Результат корректный. ($x \neq a \Rightarrow a + x = 2 + 1 = 3$)

```

[vaosina@fedora lab07]$ ./var7-2
Введите x: 2
Введите a: 1
Результат: 3

```

Рис. 4.26: Проверка работа программы для значений $x = 2, a = 1$

5 Выводы

При выполнении данной лабораторной работы я изучила команды условного и безусловного переходов, приобрела навыки написания программ с использованием переходов, а также познакомилась с назначением и структурой файла листинга.

6 Список литературы

1. Архитектура ЭВМ