

Отчет по лабораторной работе №7

Дисциплина: архитектура компьютера

Курушин Георгий Романович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Реализация переходов в NASM	8
4.2	Изучение структуры файла листинга	14
4.3	Задания для самостоятельной работы	17
5	Выводы	24
	Список литературы	25

Список иллюстраций

4.1	Создание каталога и файла для программы	8
4.2	Сохранение программы	9
4.3	Запуск программы	9
4.4	Изменение программы	10
4.5	Запуск измененной программы	11
4.6	Изменение программы	11
4.7	Проверка изменений	12
4.8	Сохранение новой программы	13
4.9	Проверка программы из листинга	14
4.10	Проверка файла листинга	15
4.11	Удаление операнда из программы	16
4.12	Просмотр ошибки в файле листинга	17
4.13	Первая программа самостоятельной работы	18
4.14	Проверка работы первой программы	20
4.15	Вторая программа самостоятельной работы	21
4.16	Проверка работы второй программы	23

Список таблиц

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

Здесь приводится описание задания в соответствии с рекомендациями методического пособия и выданным вариантом.

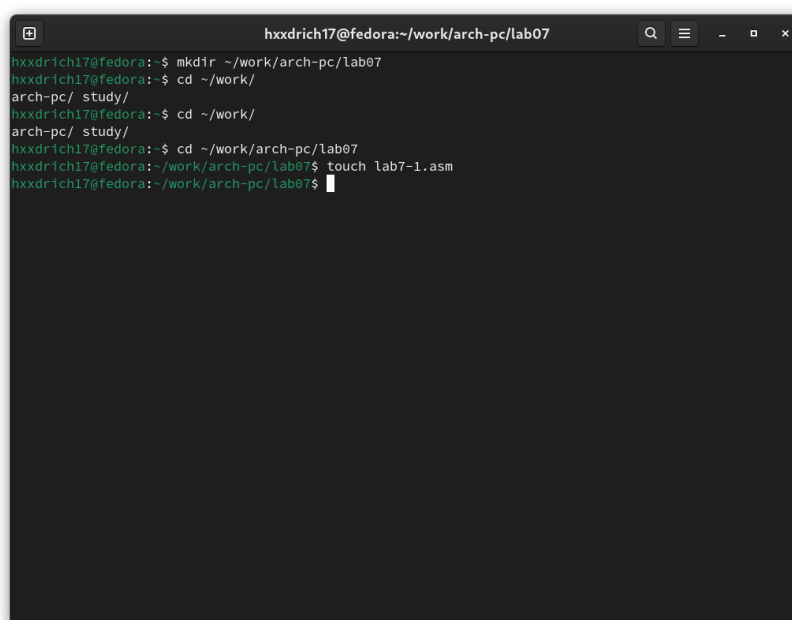
3 Теоретическое введение

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.

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

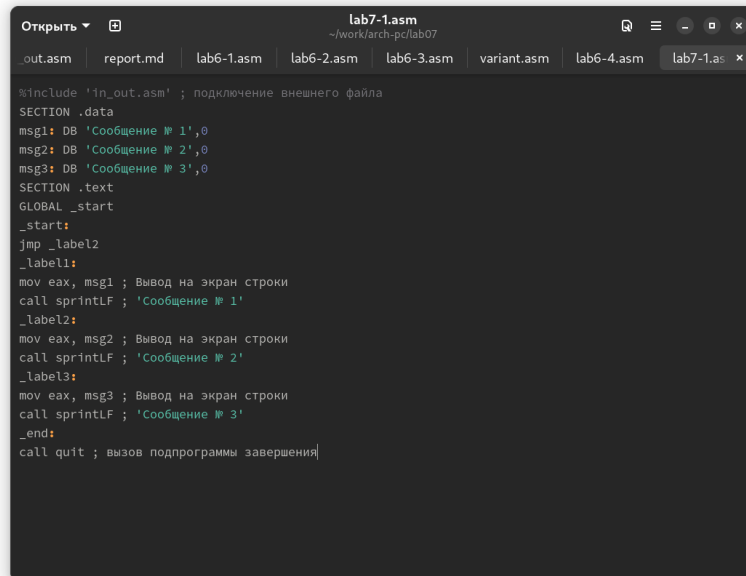
Создаю каталог для программ лабораторной работы №7 (рис. -fig. 4.1).



```
hxxdrich17@fedora:~/work/arch-pc/lab07
hxxdrich17@fedora:~$ mkdir ~/work/arch-pc/lab07
hxxdrich17@fedora:~$ cd ~/work/
arch-pc/ study/
hxxdrich17@fedora:~$ cd ~/work/
arch-pc/ study/
hxxdrich17@fedora:~$ cd ~/work/arch-pc/lab07
hxxdrich17@fedora:~/work/arch-pc/lab07$ touch lab7-1.asm
hxxdrich17@fedora:~/work/arch-pc/lab07$
```

Рис. 4.1: Создание каталога и файла для программы

Копирую код из листинга в файл будущей программы. (рис. -fig. 4.2).



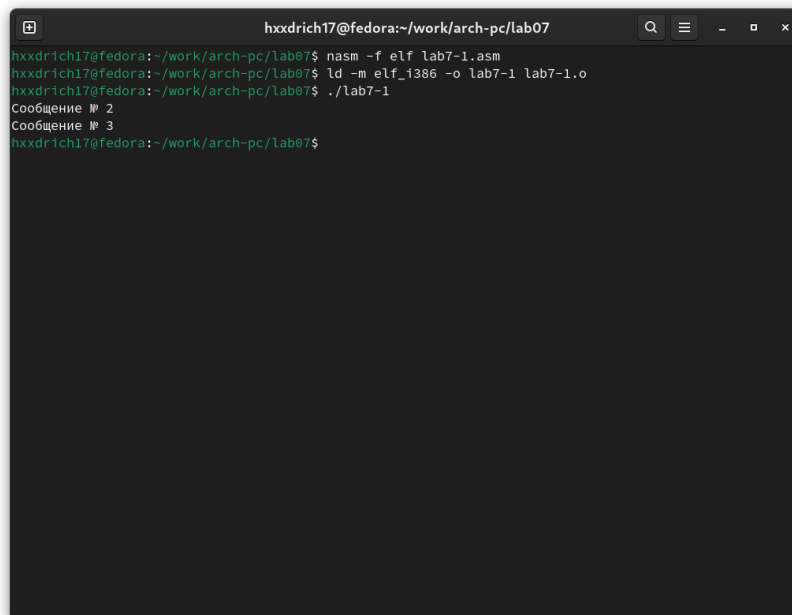
```
lab7-1.asm
~/.work/arch-pc/lab07

.out.asm | report.md | lab6-1.asm | lab6-2.asm | lab6-3.asm | variant.asm | lab6-4.asm | lab7-1.as x

%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: Сохранение программы

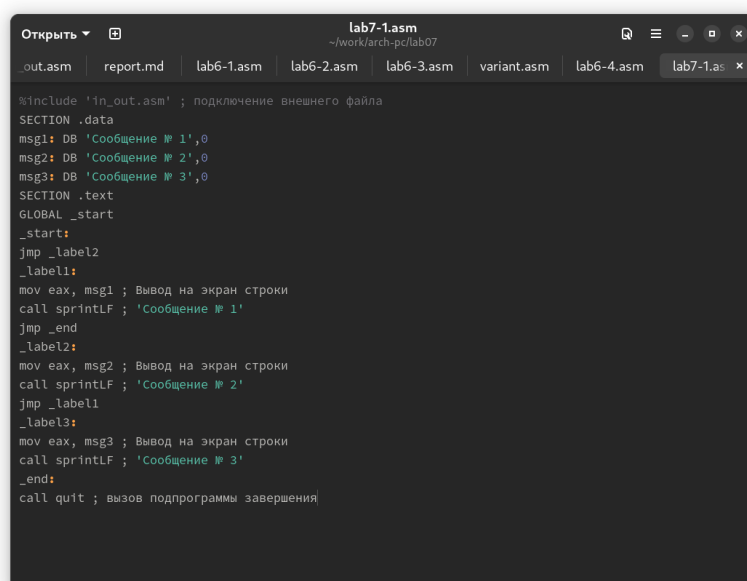
При запуске программы я убедился в том, что безусловный переход действительно изменяет порядок выполнения инструкций (рис. -fig. 4.3).



```
hxxdrich17@fedora:~/work/arch-pc/lab07
hxxdrich17@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
hxxdrich17@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
hxxdrich17@fedora:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
hxxdrich17@fedora:~/work/arch-pc/lab07$
```

Рис. 4.3: Запуск программы

Изменяю программу таким образом, чтобы поменялся порядок выполнения функций (рис. -fig. 4.4).

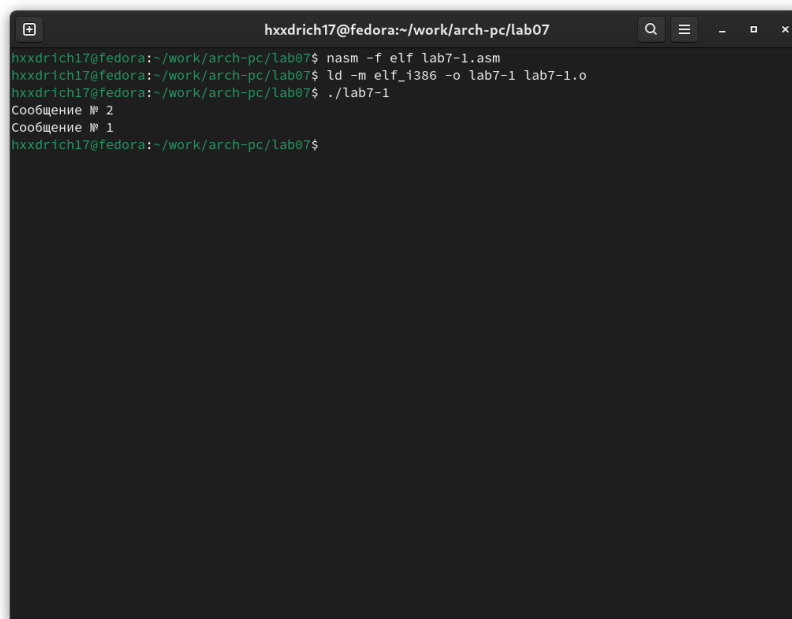


```
Открыть  lab7-1.asm
~/work/arch-pc/lab07
out.asm  report.md  lab6-1.asm  lab6-2.asm  lab6-3.asm  variant.asm  lab6-4.asm  lab7-1.as x

%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'
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'
_end:
call quit ; вызов подпрограммы завершения
```

Рис. 4.4: Изменение программы

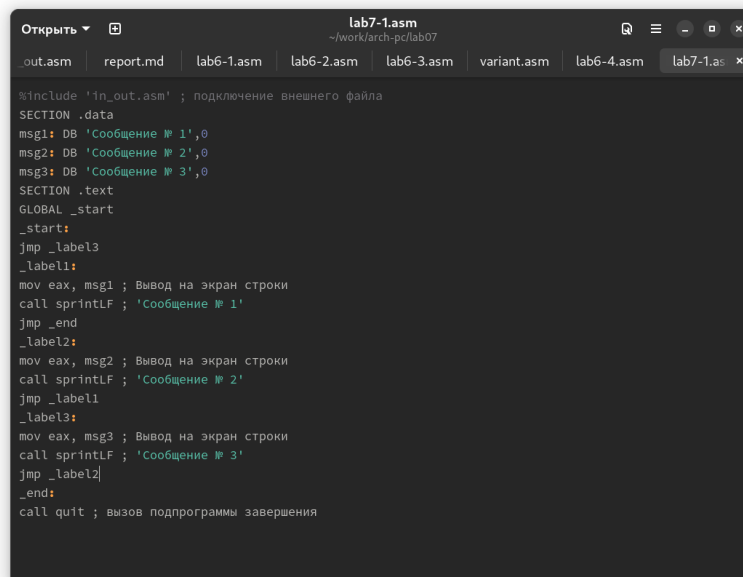
Запускаю программу и проверяю, что примененные изменения верны (рис. -fig. 4.5).



```
hxxdrich17@fedora:~/work/arch-pc/lab07
hxxdrich17@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
hxxdrich17@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
hxxdrich17@fedora:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
hxxdrich17@fedora:~/work/arch-pc/lab07$
```

Рис. 4.5: Запуск измененной программы

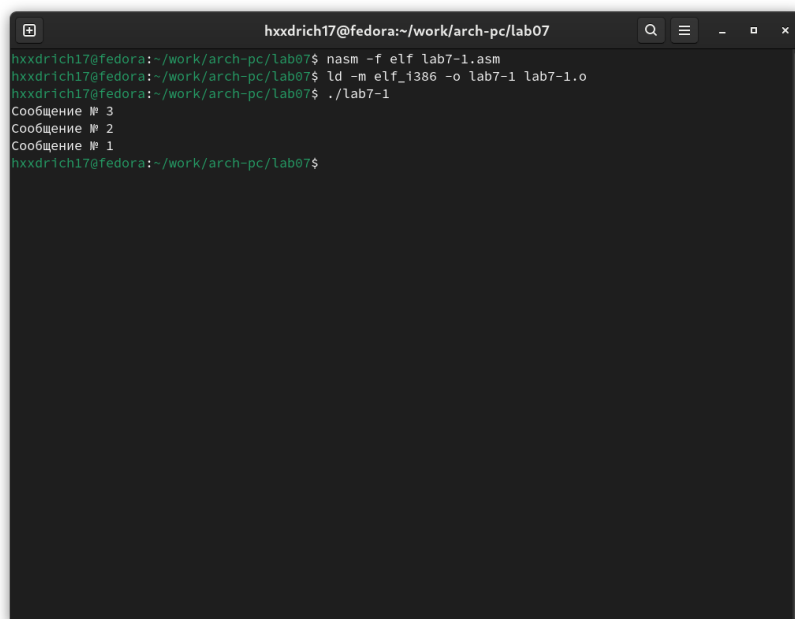
Теперь изменяю текст программы так, чтобы все три сообщения вывелись в обратном порядке (рис. -fig. 4.6).



```
lab7-1.asm
~/work/arch-pc/lab07
out.asm  report.md  lab6-1.asm  lab6-2.asm  lab6-3.asm  variant.asm  lab6-4.asm  lab7-1.as x
%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 программы

Работа выполнена корректно, программа в нужном мне порядке выводит сообщения (рис. -fig. 4.7).



```
hxxdrich17@fedora:~/work/arch-pc/lab07
hxxdrich17@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
hxxdrich17@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
hxxdrich17@fedora:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
hxxdrich17@fedora:~/work/arch-pc/lab07$
```

Рис. 4.7: Проверка изменений

Создаю новый рабочий файл и вставляю в него код из следующего листинга (рис. -fig. 4.8).

```

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

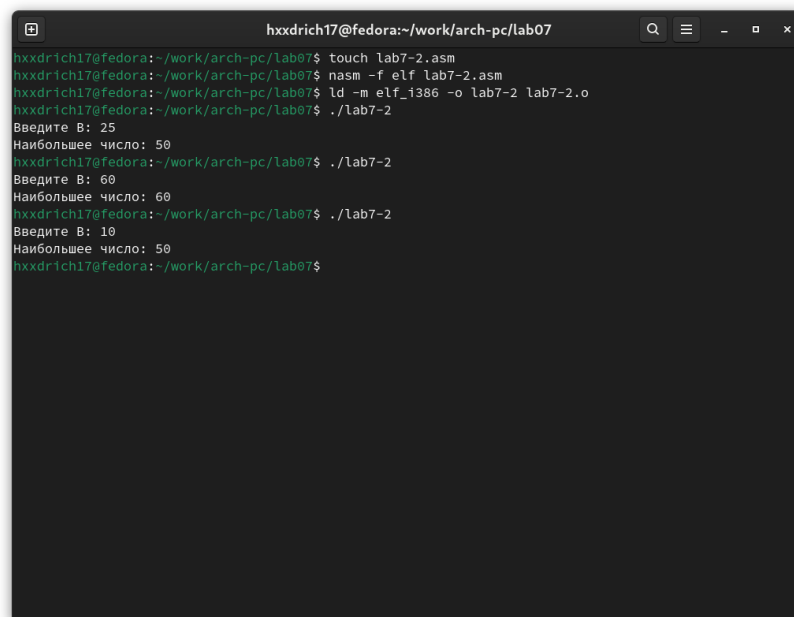
report.md lab6-1.asm lab6-2.asm lab6-3.asm variant.asm lab6-4.asm lab7-1.asm lab7-2.asm x

#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 ; Вывод подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'B'
; ----- Записываем 'A' в переменную 'max'
mov ecx,[A] ; 'rca = A'
mov [max],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'C' (как символы)
scmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', то переход на метку 'check_B',
mov ecx,[C] ; иначе 'rca = C'
mov [max],ecx ; 'max = C'
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
mov eax,[max]
call atoi ; Вывод подпрограммы перевода символа в число
mov [max],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx,[max]
scmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>B', то переход на 'fin',
mov ecx,[B] ; иначе 'rca = B'
mov [max],ecx
; ----- Вывод результата
fin:
mov eax,msg2
call sprint ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call sprintf ; Вывод 'max(A,B,C)'
call quit ; Выход

```

Рис. 4.8: Сохранение новой программы

Программа выводит значение переменной с максимальным значением, проверяя работу программы с разными входными данными (рис. -fig. 4.9).



```
hxxdrich17@fedora:~/work/arch-pc/lab07
hxxdrich17@fedora:~/work/arch-pc/lab07$ touch lab7-2.asm
hxxdrich17@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
hxxdrich17@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
hxxdrich17@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 25
Наибольшее число: 50
hxxdrich17@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 60
Наибольшее число: 60
hxxdrich17@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 10
Наибольшее число: 50
hxxdrich17@fedora:~/work/arch-pc/lab07$
```

Рис. 4.9: Проверка программы из листинга

4.2 Изучение структуры файла листинга

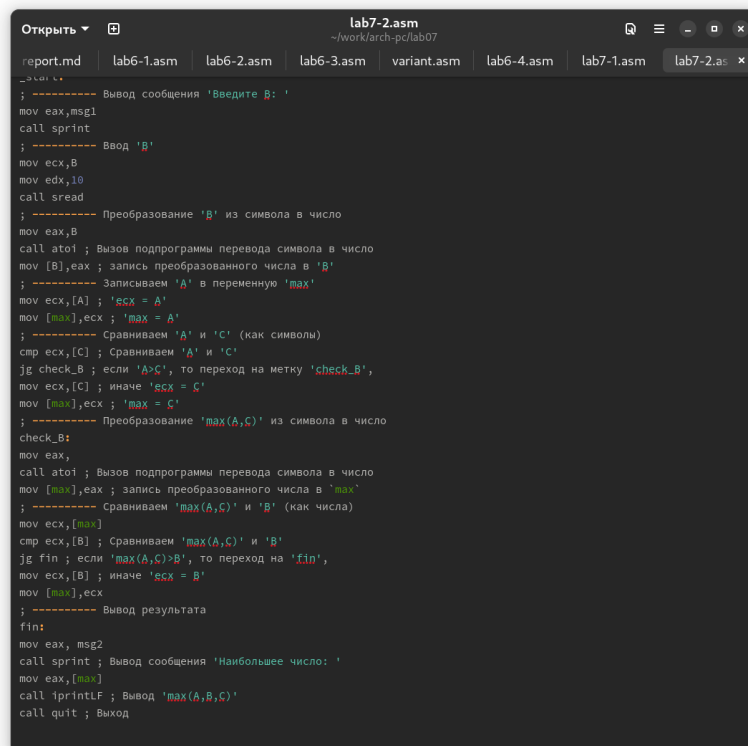
Создаю файл листинга с помощью флага `-l` команды `nasm` и открываю его с помощью текстового редактора `mcedit` (рис. -fig. 4.10).

```
lab7-2.lst [----] 0 L: 1+ 0 1/225] *(0 /14458b) 0032 0x020 [X]
1 %include 'in_out.asm'
2 ;----- slen -----
3 ; функция вычисления длины сообщения
4 slen:
5 00000000 53 <1> push ebx
6 00000001 89C3 <1> mov ebx, eax
7 <1>
8 00000003 803800 <1> nextchar:
9 00000006 7403 <1> cmp byte [eax], 0
10 00000008 40 <1> jz finished
11 00000009 EBF8 <1> inc eax
12 <1> jmp nextchar
13 <1> finished:
14 0000000B 29D8 <1> sub eax, ebx
15 0000000D 5B <1> pop ebx
16 0000000E C3 <1> ret
17 <1>
18 <1>
19 ;----- sprint -----
20 ; функция печати сообщения
21 <1> ; входные данные: mov eax, <message>
22 <1> sprint:
23 0000000F 52 <1> push edx
24 00000010 51 <1> push ecx
25 00000011 53 <1> push ebx
26 00000012 50 <1> push eax
27 00000013 E8E8FFFFFF <1> call slen
28 <1>
29 00000018 89C2 <1> mov edx, eax
30 0000001A 58 <1> pop eax
```

Рис. 4.10: Проверка файла листинга

Первое значение в файле листинга - номер строки, и он может вовсе не совпадать с номером строки изначального файла. Второе вхождение - адрес, смещение машинного кода относительно начала текущего сегмента, затем непосредственно идет сам машинный код, а заключает строку исходный текст программы с комментариями.

Удаляю один операнд из случайной инструкции, чтобы проверить поведение файла листинга в дальнейшем (рис. -fig. 4.11).



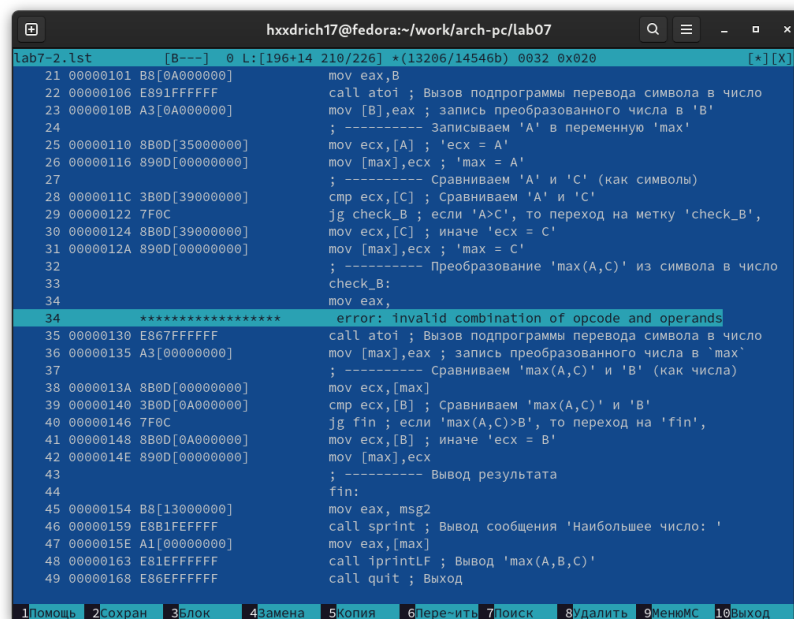
```
lab7-2.asm
~/work/arch-pc/lab07

report.md lab6-1.asm lab6-2.asm lab6-3.asm variant.asm lab6-4.asm lab7-1.asm lab7-2.asm x

; ----- 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,
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 iprintf ; Вывод 'max(A,B,C)'
call quit ; Выход
```

Рис. 4.11: Удаление операнда из программы

В новом файле листинга показывает ошибку, которая возникла при попытке трансляции файла. Никакие выходные файлы при этом помимо файла листинга не создаются. (рис. -fig. 4.12).

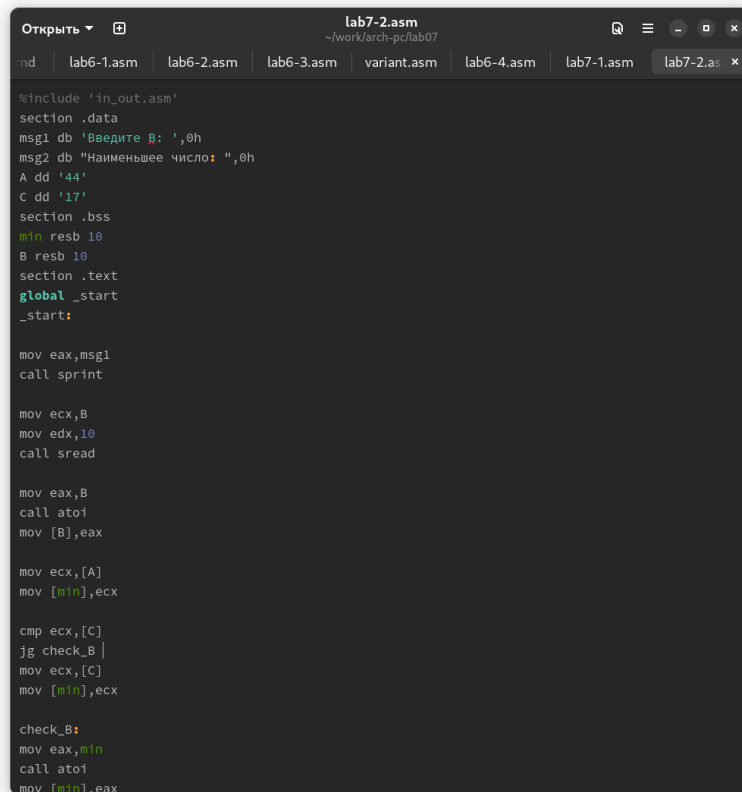


```
hxxdrich17@fedora:~/work/arch-pc/lab07
lab7-2.lst [B---] 0 L:[196+14 210/226] *(13206/14546b) 0032 0x020 [*][X]
21 00000101 B8[0A000000] mov eax,B
22 00000106 E891FFFFFF call atoi ; Вызов подпрограммы перевода символа в число
23 0000010B A3[0A000000] mov [B],eax ; запись преобразованного числа в 'B'
24 ; ----- Записываем 'A' в переменную 'max'
25 00000110 8B0D[35000000] mov ecx,[A] ; 'ecx = A'
26 00000116 890D[00000000] mov [max],ecx ; 'max = A'
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',
30 00000124 8B0D[39000000] mov ecx,[C] ; иначе 'ecx = C'
31 0000012A 890D[00000000] mov [max],ecx ; 'max = C'
32 ; ----- Преобразование 'max(A,C)' из символа в число
33 check_B:
34 mov eax,
34 ***** error: invalid combination of opcode and operands
35 00000130 E867FFFFFF call atoi ; Вызов подпрограммы перевода символа в число
36 00000135 A3[00000000] mov [max],eax ; запись преобразованного числа в 'max'
37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
38 0000013A 8B0D[00000000] mov ecx,[max]
39 00000140 3B0D[0A000000] cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
40 00000146 7F0C jg fin ; если 'max(A,C)>B', то переход на 'fin',
41 00000148 8B0D[0A000000] mov ecx,[B] ; иначе 'ecx = B'
42 0000014E 890D[00000000] mov [max],ecx
43 ; ----- Вывод результата
44 fin:
45 00000154 B8[13000000] mov eax,msg2
46 00000159 E8B1FEFFFF call sprintf ; Вывод сообщения 'Наибольшее число: '
47 0000015E A1[00000000] mov eax,[max]
48 00000163 E81EFFFFFF call iprintLF ; Вывод 'max(A,B,C)'
49 00000168 E86EFFFFFF call quit ; Выход
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пере-ить 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 4.12: Просмотр ошибки в файле листинга

4.3 Задания для самостоятельной работы

Буду использовать свой вариант - шестнадцатый - из предыдущей лабораторной работы. Возвращаю операнд к функции в программе и изменяю ее так, чтобы она выводила переменную с наименьшим значением (рис. -fig. 4.13).



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

nd | lab6-1.asm | lab6-2.asm | lab6-3.asm | variant.asm | lab6-4.asm | lab7-1.asm | lab7-2.asm x

#include 'in_out.asm'
section .data
msg1 db 'Введите B: ',0h
msg2 db "Наименьшее число: ",0h
A dd '44'
C dd '17'
section .bss
min resb 10
B resb 10
section .text
global _start
_start:

mov eax,msg1
call sprint

mov ecx,B
mov edx,10
call sread

mov eax,B
call atoi
mov [0],eax

mov ecx,[A]
mov [min],ecx

cmp ecx,[C]
jg check_B |
mov ecx,[C]
mov [min],ecx

check_B:
mov eax,min
call atoi
mov [min],eax
```

Рис. 4.13: Первая программа самостоятельной работы

Код первой программы:

```
%include 'in_out.asm'
section .data
msg1 db 'Введите B: ',0h
msg2 db "Наименьшее число: ",0h
A dd '44'
C dd '17'
section .bss
min resb 10
B resb 10
section .text
global _start
```

```

_start:

mov eax,msg1
call sprint

mov ecx,B
mov edx,10
call sread

mov eax,B
call atoi
mov [B],eax

mov ecx,[A]
mov [min],ecx

cmp ecx,[C]
jg check_B
mov ecx,[C]
mov [min],ecx

check_B:
mov eax,min
call atoi
mov [min],eax

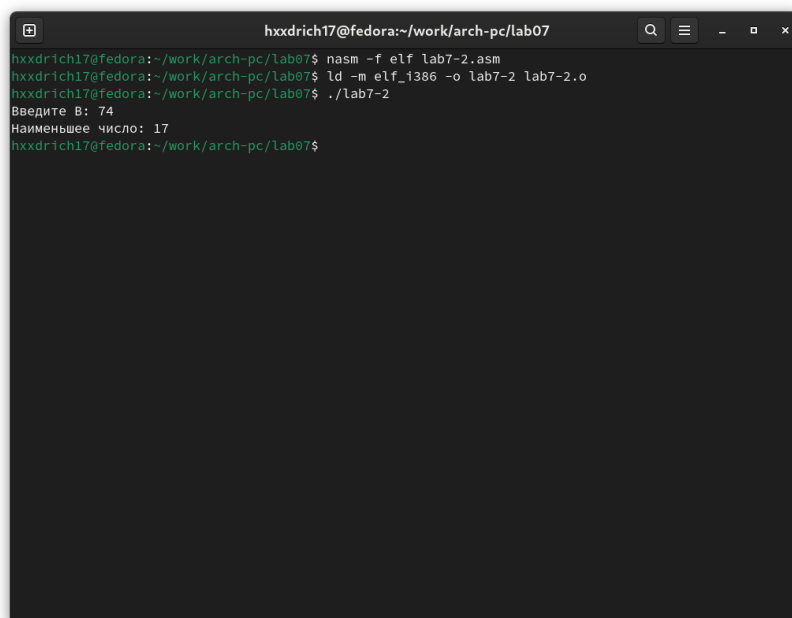
mov ecx,[min]
cmp ecx,[B]
jb fin

```

```
mov ecx, [B]
mov [min], ecx

fin:
mov eax, msg2
call sprint
mov eax, [min]
call iprintLF
call quit
```

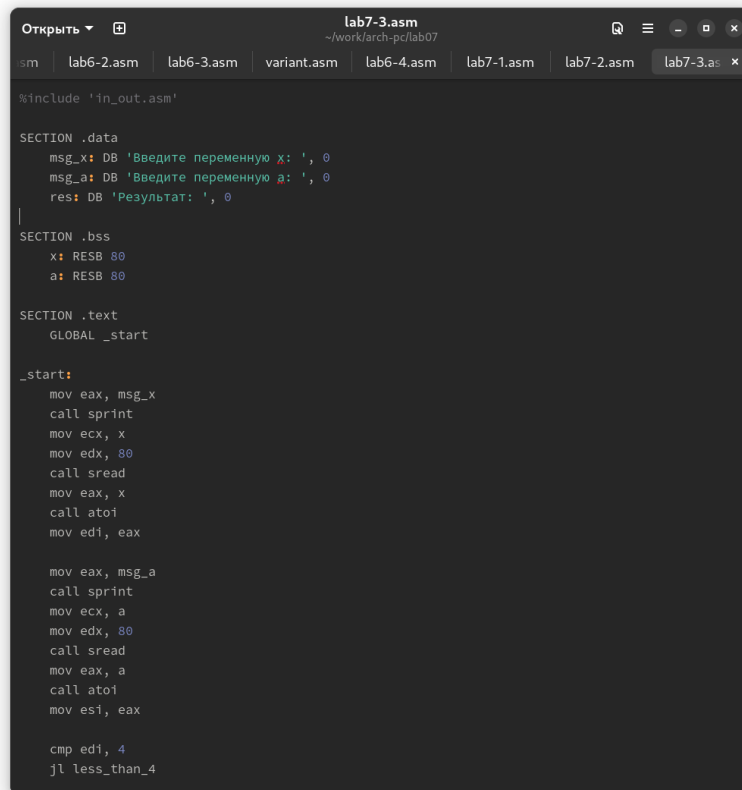
Проверяю корректность написания первой программы (рис. -fig. 4.14).



```
hxxdrich17@fedora:~/work/arch-pc/lab07
hxxdrich17@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
hxxdrich17@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
hxxdrich17@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 74
Наименьшее число: 17
hxxdrich17@fedora:~/work/arch-pc/lab07$
```

Рис. 4.14: Проверка работы первой программы

Пишу программу, которая будет вычислять значение заданной функции согласно моему варианту для введенных с клавиатуры переменных а и х (рис. -fig. 4.15).



```
lab7-3.asm
~/work/arch-pc/lab07

asm | lab6-2.asm | lab6-3.asm | variant.asm | lab6-4.asm | lab7-1.asm | lab7-2.asm | lab7-3.asm x

#include 'in_out.asm'

SECTION .data
msg_x: DB 'Введите переменную x: ', 0
msg_a: DB 'Введите переменную a: ', 0
res: DB 'Результат: ', 0

SECTION .bss
x: RESB 80
a: RESB 80

SECTION .text
GLOBAL _start

_start:
    mov eax, msg_x
    call sprint
    mov ecx, x
    mov edx, 80
    call sread
    mov eax, x
    call atoi
    mov edi, eax

    mov eax, msg_a
    call sprint
    mov ecx, a
    mov edx, 80
    call sread
    mov eax, a
    call atoi
    mov esi, eax

    cmp edi, 4
    jl less_than_4
```

Рис. 4.15: Вторая программа самостоятельной работы

Код второй программы:

```
%include 'in_out.asm'

SECTION .data
    msg_x: DB 'Введите переменную x: ', 0
    msg_a: DB 'Введите переменную a: ', 0
    res: DB 'Результат: ', 0

SECTION .bss
    x: RESB 80
    a: RESB 80
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
    mov eax, msg_x
```

```
    call sprint
```

```
    mov ecx, x
```

```
    mov edx, 80
```

```
    call sread
```

```
    mov eax, x
```

```
    call atoi
```

```
    mov edi, eax
```

```
    mov eax, msg_a
```

```
    call sprint
```

```
    mov ecx, a
```

```
    mov edx, 80
```

```
    call sread
```

```
    mov eax, a
```

```
    call atoi
```

```
    mov esi, eax
```

```
    cmp edi, 4
```

```
    jl less_than_4
```

```
    ; Ветка x >= 4
```

```
    mov eax, edi
```

```
    imul eax, esi
```

```
    jmp print_result
```

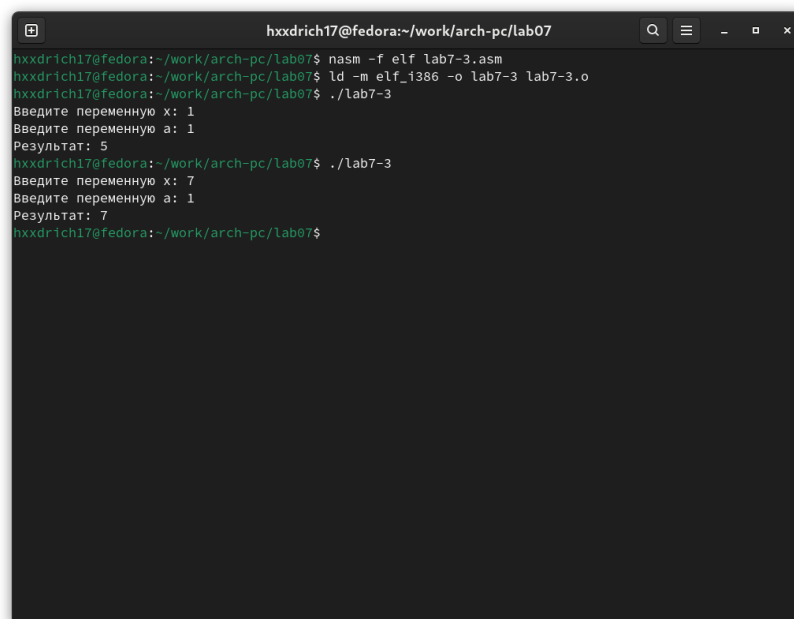
```

less_than_4:
    mov eax, edi
    add eax, 4

print_result:
    mov edi, eax
    mov eax, res
    call sprint
    mov eax, edi
    call iprintLF
    call quit

```

Транслирую и компоную файл, запускаю и проверяю работу программы для различных значений а и х (рис. -fig. 4.16).



```

hxxdrich17@fedora:~/work/arch-pc/lab07
hxxdrich17@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-3.asm
hxxdrich17@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
hxxdrich17@fedora:~/work/arch-pc/lab07$ ./lab7-3
Введите переменную x: 1
Введите переменную a: 1
Результат: 5
hxxdrich17@fedora:~/work/arch-pc/lab07$ ./lab7-3
Введите переменную x: 7
Введите переменную a: 1
Результат: 7
hxxdrich17@fedora:~/work/arch-pc/lab07$

```

Рис. 4.16: Проверка работы второй программы

5 Выводы

При выполнении лабораторной работы я изучил команды условных и безусловных переходов, а также приобрел навыки написания программ с использованием переходов, познакомился с назначением и структурой файлов листинга.

Список литературы

1. Курс на ТУИС
2. Лабораторная работа №7
3. Программирование на языке ассемблера NASM Столяров А. В.