

Отчет по лабораторной работе №7

Дисциплина: архитектура компьютера

Панина Жанна Валерьевна

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	16
5	Выводы	23
	Список литературы	24

Список иллюстраций

4.1	Создание каталога и файла для программы	8
4.2	Сохранение программы	9
4.3	Запуск исполняемого файла	9
4.4	Изменение программы	10
4.5	Запуск изменённой программы	11
4.6	Изменение программы	11
4.7	Запуск изменённого файла	12
4.8	Сохранение новой программы	13
4.9	Проверка программы из листинга	13
4.10	Проверка файла листинга	14
4.11	Удаление операнда из программы	15
4.12	Просмотр ошибки в файле листинга	16
4.13	Первая программа самостоятельной работы	17
4.14	Проверка работы первой программы	19
4.15	Вторая программа самостоятельной работы	20
4.16	Проверка работы второй программы	22

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

1. Реализация переходов в NASM
2. Изучение структуры файлов листинга
3. Самостоятельное написание программ по материалам лабораторной работы

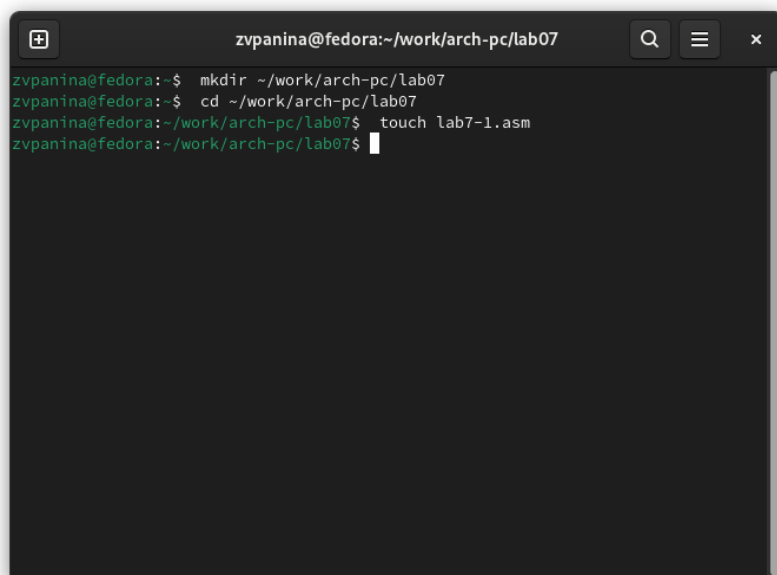
3 Теоретическое введение

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.

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

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

A terminal window with a dark background and light text. The title bar shows the user 'zvpanina' on a 'fedora' machine, in the directory '~/work/arch-pc/lab07'. The terminal contains four lines of commands and their outputs: 1. 'mkdir ~/work/arch-pc/lab07' followed by a new line. 2. 'cd ~/work/arch-pc/lab07' followed by a new line. 3. 'touch lab7-1.asm' followed by a new line. 4. The prompt 'zvpanina@fedora:~/work/arch-pc/lab07\$' with a cursor. The window has standard Linux window controls (minimize, maximize, close) and a search icon in the top right.

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



```
%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
call printf
; 'Сообщение № 1'

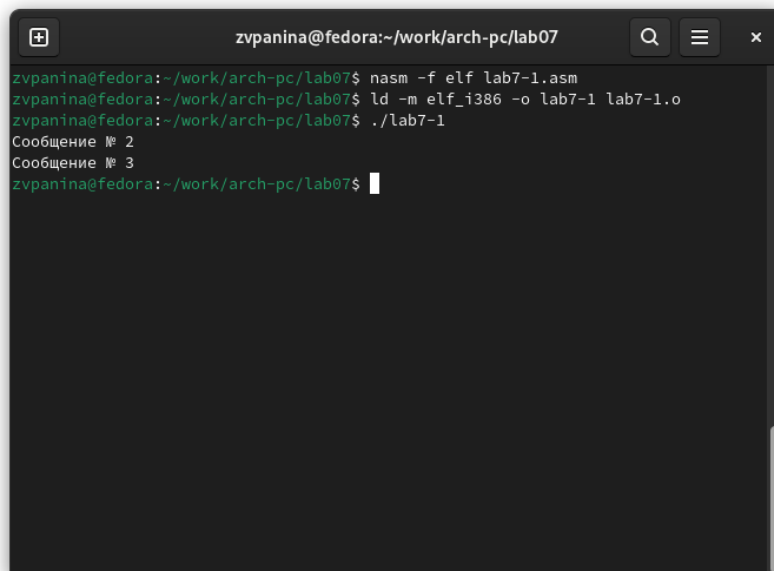
_label2:
mov eax, msg2
call printf
; 'Сообщение № 2'

_label3:
mov eax, msg3
call printf
; 'Сообщение № 3'

_end:
call _exit
; вызов подпрограммы завершения
```

Рис. 4.2: Сохранение программы

Создаю исполняемый файл и запускаю его. Вижу, что использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения (рис. -fig. 4.3).



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

Рис. 4.3: Запуск исполняемого файла

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

```
%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 sax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
jmp _end

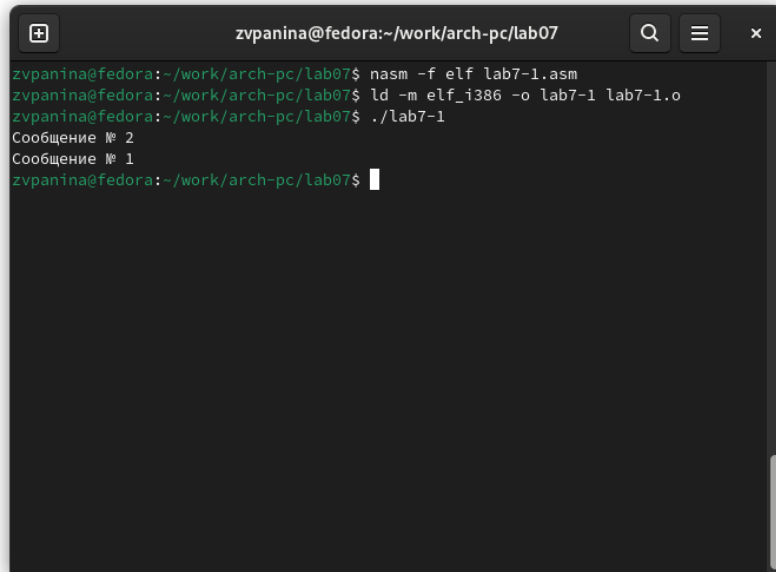
_label2:
mov sax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
jmp _label1

_label3:
mov sax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'

_end:
call _exit ; вызов подпрограммы завершения
```

Рис. 4.4: Изменение программы

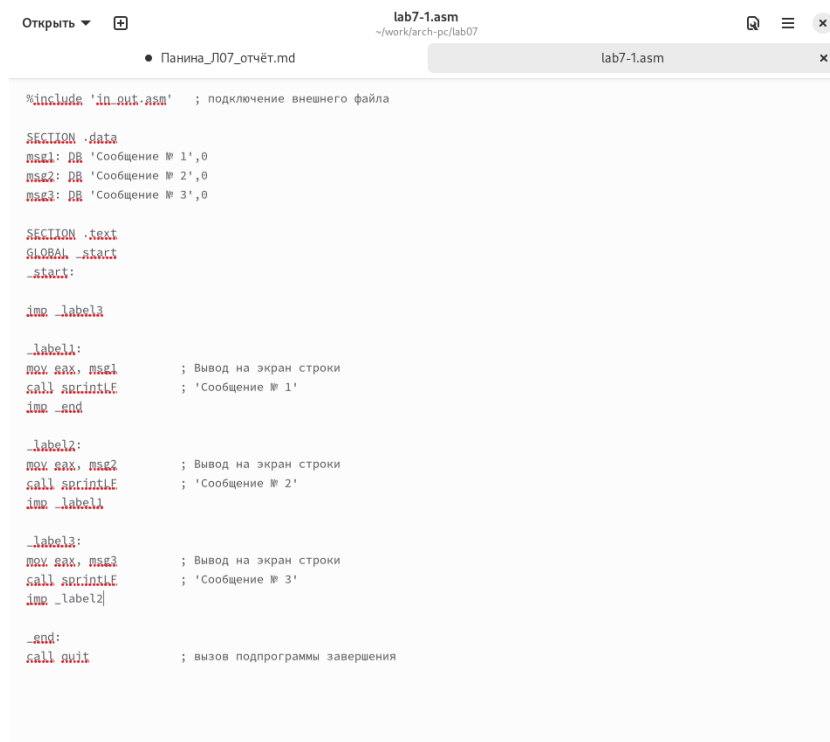
Запускаю программу и проверяю, что примененные изменения верны (рис. -fig. 4.5).



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

Рис. 4.5: Запуск изменённой 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

%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 print_string  ; 'Сообщение № 1'
jmp _end

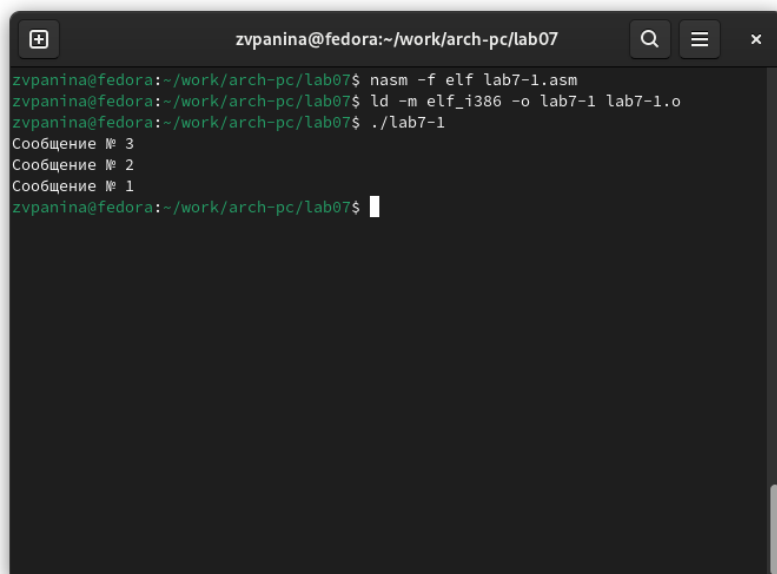
_label2:
mov eax, msg2      ; Вывод на экран строки
call print_string  ; 'Сообщение № 2'
jmp _label1

_label3:
mov eax, msg3      ; Вывод на экран строки
call print_string  ; '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).



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

Рис. 4.7: Запуск изменённого файла

Создаю файл lab7-2.asm и вставляю в него код из листинга 7.3 (рис. -fig. 4.8).

```

%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 _write
;----- Ввод 'B'
mov ecx,B
mov edx,10
call _read
;----- 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'
je 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,[max]
call atoi ; Вызов подпрограммы перевода символа в число

```

Рис. 4.8: Сохранение новой программы

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

```

zvpanina@fedora:~/work/arch-pc/lab07
zvpanina@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
zvpanina@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
zvpanina@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 15
Наибольшее число: 50
zvpanina@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 70
Наибольшее число: 70
zvpanina@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 0
Наибольшее число: 50
zvpanina@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 mousepad (рис. -fig. 4.10).

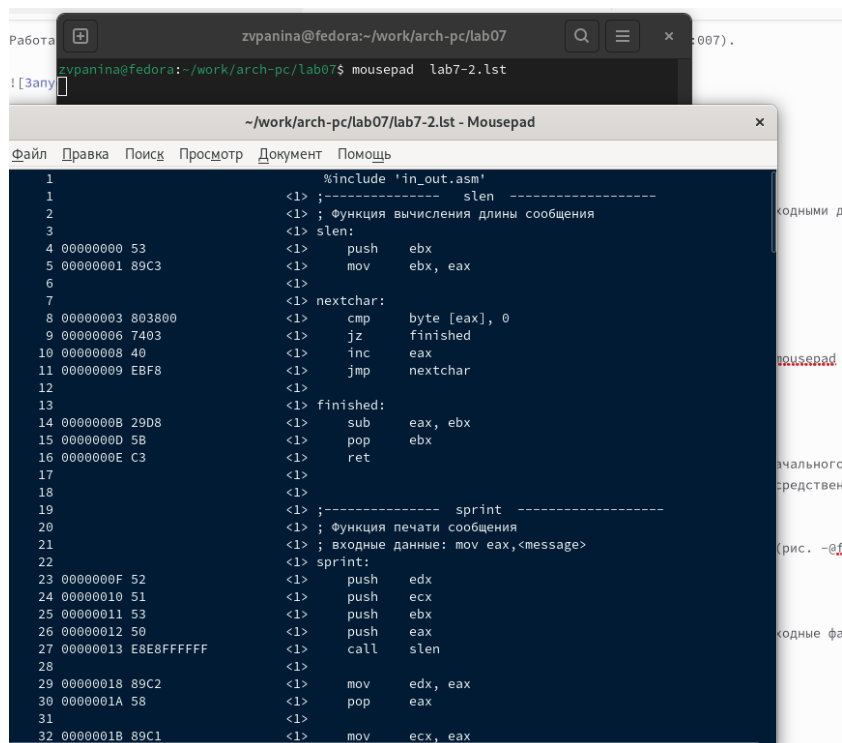
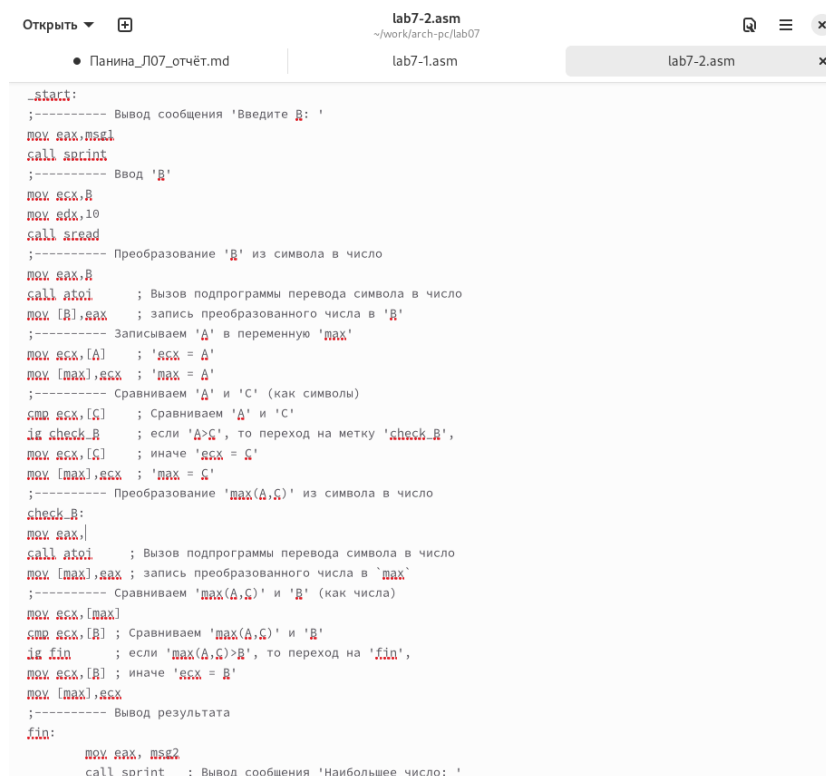


Рис. 4.10: Проверка файла листинга

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).



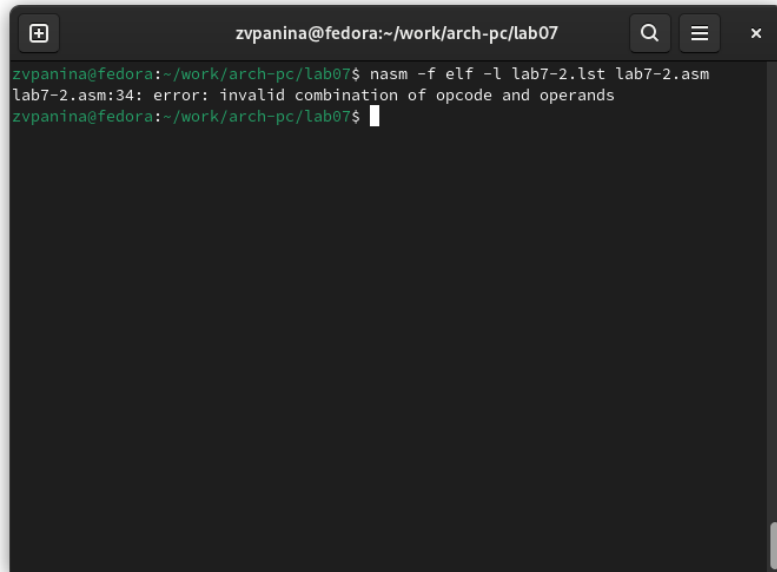
```
lab7-2.asm
~/work/arch-pc/lab07

Панина_ЛЮ7_отчёт.md | lab7-1.asm | lab7-2.asm x

_start:
;----- Вывод сообщения 'Введите R: '
mov eax, msg1
call printf
;----- Ввод 'R'
mov ecx, R
mov edx, 10
call scanf
;----- Преобразование 'R' из символа в число
mov eax, R
call atoi ; Вызов подпрограммы перевода символа в число
mov [R], eax ; запись преобразованного числа в 'R'
;----- 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'
je check_R ; если 'A>C', то переход на метку 'check_R',
mov ecx, [C] ; иначе 'ecx = C'
mov [max], ecx ; 'max = C'
;----- Преобразование 'max(A,C)' из символа в число
check_R:
mov eax, [R]
call atoi ; Вызов подпрограммы перевода символа в число
mov [max], eax ; запись преобразованного числа в 'max'
;----- Сравниваем 'max(A,C)' и 'R' (как числа)
mov ecx, [max]
cmp ecx, [R] ; Сравниваем 'max(A,C)' и 'R'
je fin ; если 'max(A,C)>R', то переход на 'fin',
mov ecx, [R] ; иначе 'ecx = R'
mov [max], ecx
;----- Вывод результата
fin:
mov eax, msg2
call printf ; Вывод сообщения 'Наибольшее число: '
```

Рис. 4.11: Удаление операнда из программы

Выполняю трансляцию с получением файла листинга. 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).

A terminal window with a dark background. The title bar shows the user 'zvpanina' on a 'fedora' machine in the directory '~/work/arch-pc/lab07'. The terminal text shows the command 'nasm -f elf -l lab7-2.lst lab7-2.asm' being executed, followed by an error message: 'lab7-2.asm:34: error: invalid combination of opcode and operands'. The prompt returns to the shell.

```
zvpanina@fedora:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:34: error: invalid combination of opcode and operands
zvpanina@fedora:~/work/arch-pc/lab07$
```

Рис. 4.12: Просмотр ошибки в файле листинга

4.3 Задания для самостоятельной работы

1. Буду использовать значения переменных из варианта 11, который выпал мне при выполнении прошлой 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

Панина_Л07_отчёт.md | 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 '21'
C dd '34'

SECTION .bss
min 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 'min'
mov ecx, [A] ; 'ecx = A'
mov [min], ecx ; 'min = A'
;----- Сравниваем 'A' и 'C' (как символы)
cmp ecx, [C] ; Сравниваем 'A' и 'C'
jl check_B ; если 'A < C', то переход на метку 'check_B'
mov ecx, [C] ; иначе 'ecx = C'
mov [min], ecx ; 'min = C'
;----- Преобразование 'min' из символа в число
```

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 '21'
C dd '34'

SECTION .bss
min 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 'min'
mov ecx, [A]   ; 'ecx = A'
mov [min], ecx ; 'min = A'
;----- Сравниваем 'A' и 'C' (как символы)
cmp ecx, [C]   ; Сравниваем 'A' и 'C'
jlt check_B    ; если 'A<C', то переход на метку 'check_B'
mov ecx, [C]   ; иначе 'ecx = C'
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'

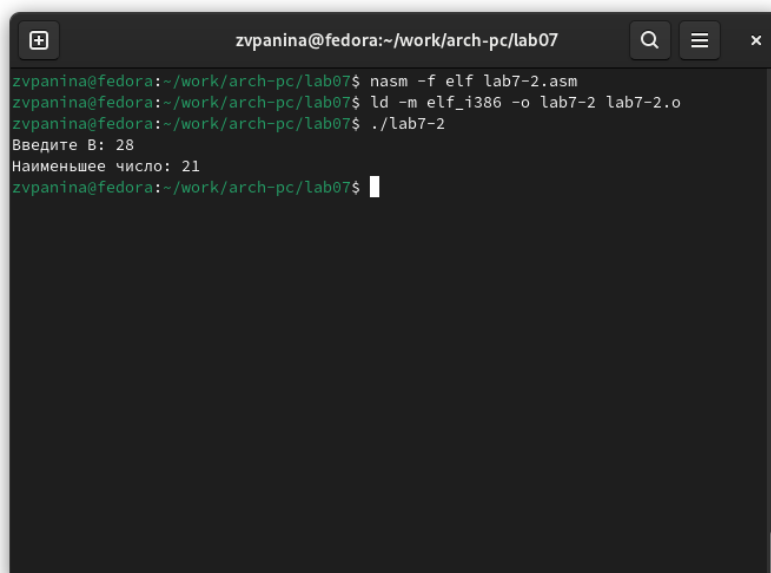
```

```

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, msg2
call sprint     ; Вывод сообщения 'Наибольшее число: '
mov eax, [min]
call iprintLF   ; Вывод 'max(A,B,C)'
call quit       ; Выход

```

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



```

zvpanina@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
zvpanina@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
zvpanina@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 28
Наименьшее число: 21
zvpanina@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).

```

%include 'in_out.asm'
SECTION .data
msg1: DB 'Введите значение переменной x: ', 0
msg2: DB 'Введите значение переменной a: ', 0
msg3: DB 'Результат: ', 0

SECTION .bss
x: RESB 80
a: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax, msg1
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
mov edi, eax

mov eax, msg2
call sprint
mov ecx, a
mov edx, 80
call sread
mov eax, a
call atoi
mov esi, eax

```

Рис. 4.15: Вторая программа самостоятельной работы

Код второй программы:

```

#include 'in_out.asm'
SECTION .data
msg1: DB 'Введите значение переменной x: ', 0
msg2: DB 'Введите значение переменной a: ', 0
msg3: DB 'Результат: ', 0

SECTION .bss
x: RESB 80
a: RESB 80

SECTION .text
GLOBAL _start
_start:

```

```
mov eax, msg1
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
mov edi, eax
```

```
mov eax, msg2
call sprint
mov ecx, a
mov edx, 80
call sread
mov eax, a
call atoi
mov esi, eax
```

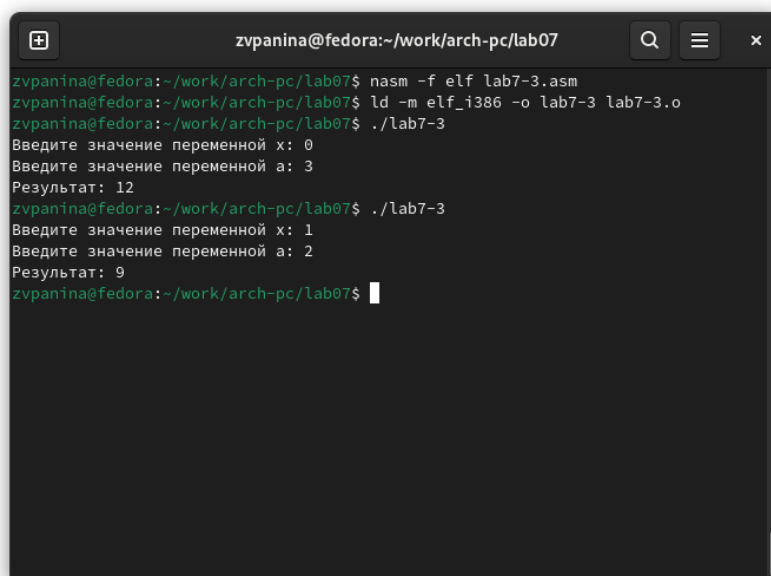
```
cmp esi, 0
jne add_values
mov eax, esi
mov ebx, 4
mul ebx
jmp print_result
```

```
add_values:
mov eax, esi
mov ebx, 4
```

```
mul ebx
add eax, edi

print_result:
mov edi, eax
mov eax, msg3
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).



```
zvpanina@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-3.asm
zvpanina@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
zvpanina@fedora:~/work/arch-pc/lab07$ ./lab7-3
Введите значение переменной x: 0
Введите значение переменной a: 3
Результат: 12
zvpanina@fedora:~/work/arch-pc/lab07$ ./lab7-3
Введите значение переменной x: 1
Введите значение переменной a: 2
Результат: 9
zvpanina@fedora:~/work/arch-pc/lab07$
```

Рис. 4.16: Проверка работы второй программы

5 Выводы

При выполнении лабораторной работы я изучила команды условных и безусловных переходов, а также приобрела навыки написания программ с использованием переходов, познакомилась 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.