

Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютеров и операционные системы

Иванова Мария Александровна

Содержание

1 Цель работы

Изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

2 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`. 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.

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.

3 Выполнение лабораторной работы

3.1 Реализация переходов в NASM

Создаю каталог для программ лабораторной работы № 7, перехожу в него и создаю файл `lab7-1.asm`. (рис. 3.1)

```
[maivanova@fedora labs]$ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-
pc/labs/lab07
[maivanova@fedora lab07]$ touch lab7-1.asm
```

Рис 3.1: Создание файлов для лабораторной работы

Ввожу в файл lab7-1.asm текст программы из листинга 7.1. (рис. 3.2).

```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07/lab7-1.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь

%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 3.2: Ввод текста программы из листинга 7.1

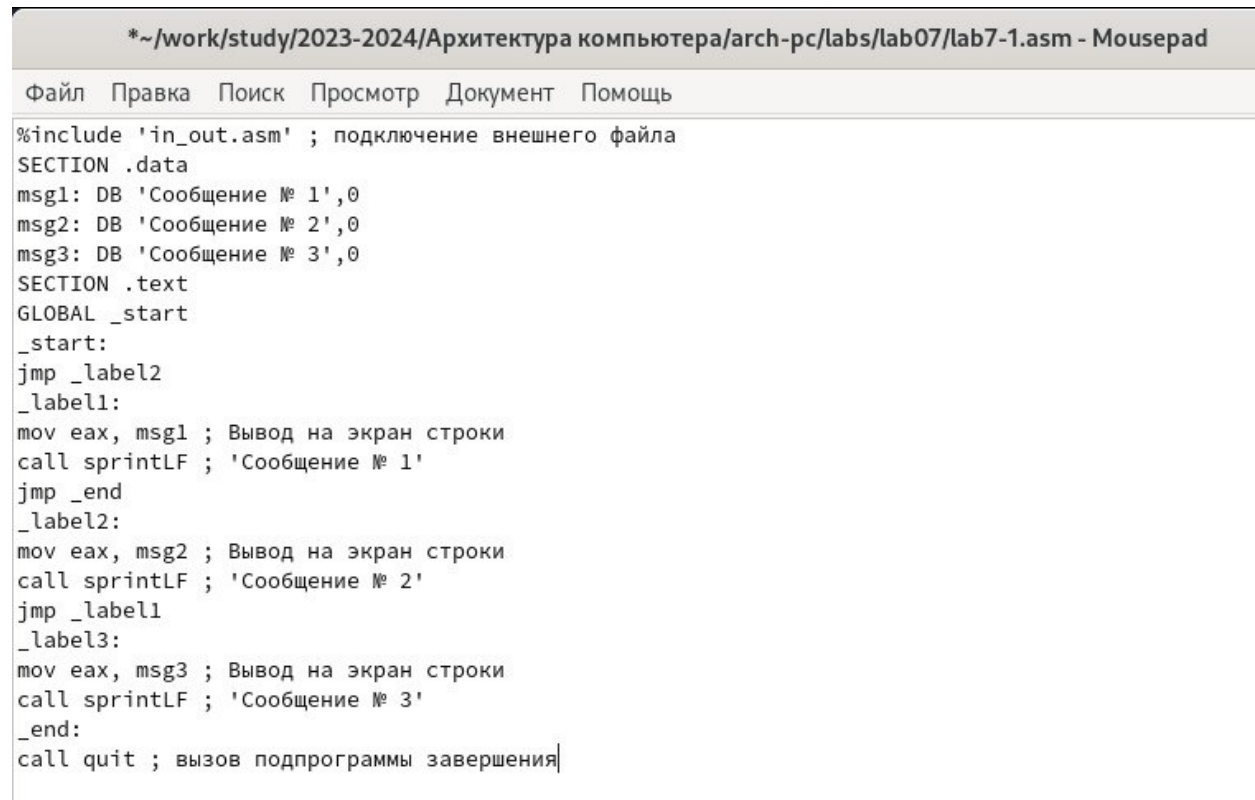
Создаю исполняемый файл и запускаю его. (рис.3.3).

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

Рис 3.3: Запуск программного кода

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

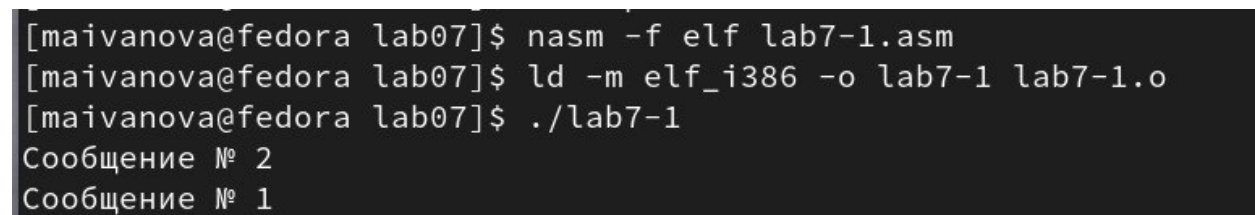
Изменяю программу таким образом, чтобы она выводила сначала 'Сообщение № 2', потом 'Сообщение № 1' и завершала работу. Для этого изменяю текст программы в соответствии с листингом 7.2. (рис.3.4).



```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07/lab7-1.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь
%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 3.4: Изменение текста программы

Создаю исполняемый файл и проверяю его работу. (рис. 3.5).



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

Рис 3.5: Создание исполняемого файла

Затем изменяю текст программы, добавив в начале программы `jmp _label3`, `jmp _label2` в конце метки `jmp _label3`, `jmp _label1` добавляю в конце метки `jmp _label2`, и добавляю `jmp _end` в конце метки `jmp _label1`, (рис. 3.6).

```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07/lab7-1.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь
%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 3.6: Изменение текста программы

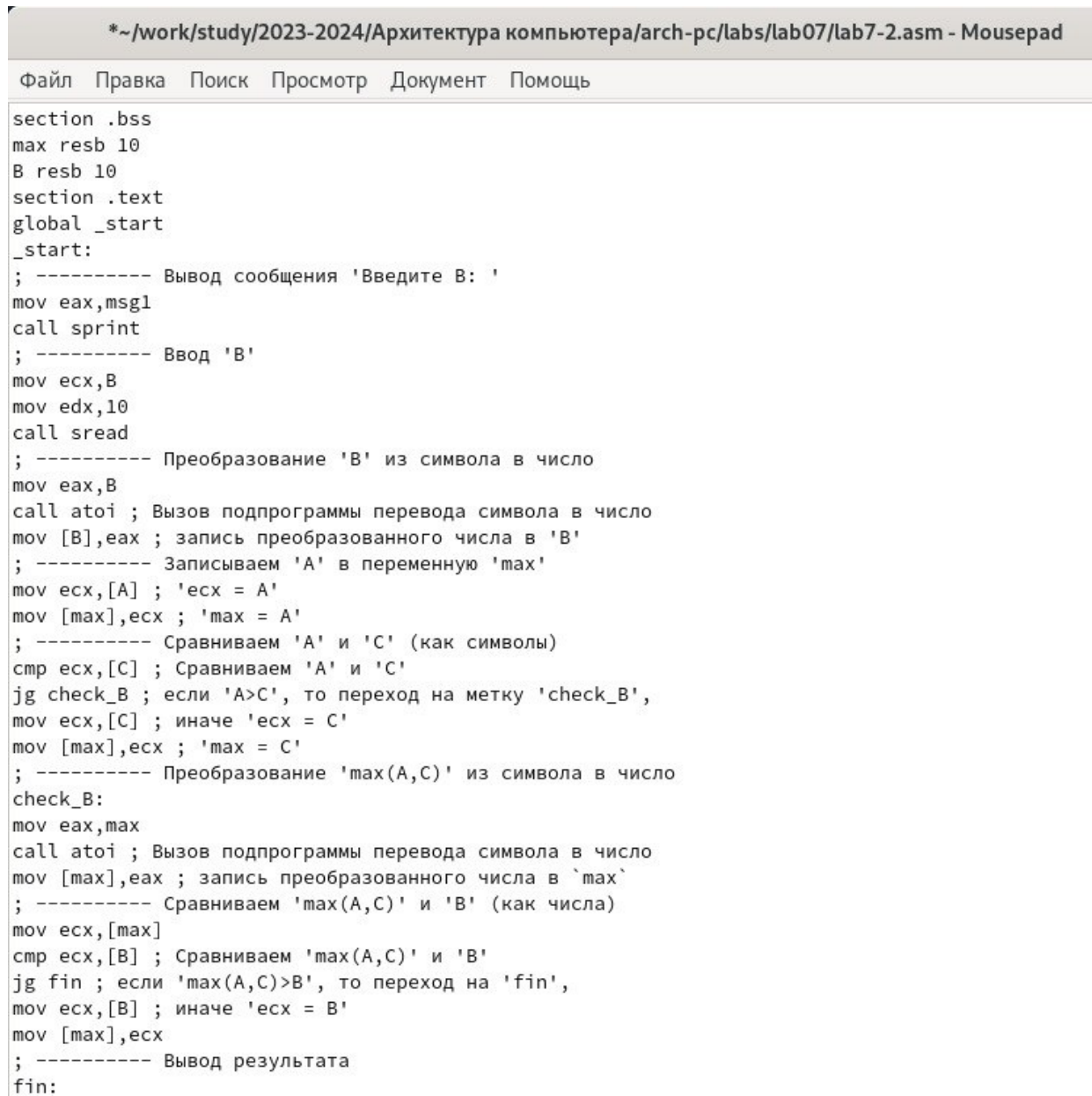
чтобы вывод программы был следующим: (рис.3.7).

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

Рис 3.7: Вывод программы

Рассмотрим программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: A, B и C. Значения для A и C задаются в программе, значение B вводится с клавиатуры.

Создаю файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. Текст программы из листинга 7.3 ввожу в lab7-2.asm. (рис.3.8).



```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07/lab7-2.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь

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'
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,max
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:
```

Рис 3.8: Ввод текста программы из листинга 7.3

Создаю исполняемый файл и проверьте его работу. (рис.3.9).

```
[maivanova@fedora lab07]$ nasm -f elf lab7-2.asm
[maivanova@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[maivanova@fedora lab07]$ ./lab7-2
Введите В: 87
Наибольшее число: 87
```

Рис 3.9: Проверка работы файла

Файл работает корректно.

3.2 Изучение структуры файлы листинга

Создаю файл листинга для программы из файла lab7-2.asm. Открываю файл листинга lab7-2.lst с помощью текстового редактора и внимательно изучаю его формат и содержимое. (рис. 3.10).

```
lab7-2.lst [----] 0 L: [ 1+ 0 1/225] *(0 /14458b) 0032 0x020 [*][X]
1 %include 'in_out.asm'
2 <1> ;----- slen -----
3 <1> ; Функция вычисления длины сообщения
4 <1> slen:.....
5 00000000 53 <1> push ebx.....
6 00000001 89C3 <1> mov ebx, eax.....
7 <1> .....
8 <1> nextchar:.....
9 00000003 803800 <1> cmp byte [eax], 0...
10 00000006 7403 <1> jz finished.....
11 00000008 40 <1> inc eax.....
12 00000009 EBF8 <1> jmp nextchar.....
13 <1> .....
14 <1> finished:
15 0000000B 29D8 <1> sub eax, ebx
16 0000000D 5B <1> pop ebx.....
17 0000000E C3 <1> ret.....
18 <1> .....
19 <1> ;----- sprint -----
20 <1> ; Функция печати сообщения
21 <1> ; входные данные: mov eax,<message>
```

1Помощь 2Сох~ть 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Уда~ть 9МенюМС 10Выход

Рис 3.10: Изучение файла листинга

В представленных трех строчках содержатся следующие данные: (рис 3.11).

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

Рис 3.11: Выбранные строки файла

“2” - номер строки кода, “; Функция вычисления длинны 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.

Открываю файл с программой lab7-2.asm и в выбранной мной инструкции с двумя операндами удаляю выделенный операнд. (рис. 3.12).

```
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'
```

Рис 3.12: Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга. (рис. 3.13).

```
[maivanova@fedora lab07]$ mousepad lab7-2.asm
[maivanova@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands
```

Рис 3.13: Получение 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.

4 Задания для самостоятельной работы

Пишу программу нахождения наименьшей из 3 целочисленных переменных a, b и c. 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 и 54. (рис. 4.1).

```
~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07/lab7-3.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь

%include 'in_out.asm'
section .data
msg db "Наименьшее число: ",0h
A dd '32'
B dd '6'
C dd '54'
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,[B] ; Сравниваем 'A' и 'C'
jl check_C ; если 'A<C', то переход на метку 'check_B',
mov ecx,[B] ; иначе 'ecx = C'
mov [min],ecx ; 'min = C'
; ----- Преобразование 'min(A,C)' из символа в число
check_C:
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
```

Рис 4.1: Написание программы

Создаю исполняемый файл и проверяю его работу, подставляя необходимые значения. (рис. 4.2).

```
[maivanova@fedora lab07]$ nasm -f elf lab7-3.asm
[maivanova@fedora lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o
[maivanova@fedora lab07]$ ./lab7-3
Наименьшее число: 6
```

Рис 4.2: Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```
%include 'in_out.asm'

section .data
msg db "Наименьшее число: ",0h
A dd '32'
B dd '6'
C dd '54'

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,[B] ; Сравниваем 'A' и 'C'
jl check_C ; если 'A<C', то переход на метку 'check_B',
mov ecx,[B] ; иначе 'ecx = C'
mov [min],ecx ; 'min = C'
; ----- Преобразование 'min(A,C)' из символа в число
check_C:
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,[C] ; Сравниваем 'min(A,C)' и 'B'
jl fin ; если 'min(A,C)<B', то переход на 'fin',
mov ecx,[C] ; 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

```

Пишу программу, которая для введенных с клавиатуры значений x и a вычисляет значение и выводит результат вычислений заданной для моего варианта функции $f(x)$:

$a + 10$, если $x < a$

$x + 10$, если $x \geq a$

(рис. 4.3).

```
~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07/lab7-4.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь

%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]
```

Рис 4.3: Написание программы

Создаю исполняемый файл и проверяю его работу для значений x и a соответственно: (2;3), (4;2). (рис. 4.4).

```
[maivanova@fedora lab07]$ mousepad lab7-3.asm
[maivanova@fedora lab07]$ nasm -f elf lab7-3.asm
[maivanova@fedora lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o
[maivanova@fedora lab07]$ ./lab7-3
Наименьшее число: 6
[maivanova@fedora lab07]$ mousepad lab7-4.asm
[maivanova@fedora lab07]$ nasm -f elf lab7-4.asm
[maivanova@fedora lab07]$ ld -m elf_i386 -o lab7-4 lab7-4.o
[maivanova@fedora lab07]$ ./lab7-4
Введите x:2
Введите a:3
Ответ:13
[maivanova@fedora lab07]$ ./lab7-4
Введите x:4
Введите a:2
Ответ:14
```

Рис 4.4: Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```
%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
```

5 Выводы

По итогам 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.

6 Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).