

Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютеров и операционные системы

Светцова Анна Дмитриевна

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 Задание

1. Реализация переходов в NASM.
2. Изучение структуры файлы листинга.
3. Задания 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`. Инструкция `str` 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.

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.1).

```
adsvettsova@bannpc ~$ mkdir ~/work/study/2023-2024/"Computer architecture"/arh-pc/labs/lab07/lab7
adsvettsova@bannpc ~$ cd ~/work/study/2023-2024/"Computer architecture"/arh-pc/labs/lab07/lab7
adsvettsova@bannpc ~/w/s/2/C/arh-pc/l/lab07/lab7 master +1 128 78 $ touch lab7-1.asm
adsvettsova@bannpc ~/w/s/2/C/arh-pc/l/lab07/lab7 master +1 128 79 $ 20:33:35
```

рис.1 Создание файлов для лабораторной работы

Ввожу в файл `lab7-1.asm` текст программы из листинга 7.1. (рис. 2).

```
lab7-1.asm
1
2 SECTION .data
3 msg1 DB 'Сообщение № 1',0
4 msg2 DB 'Сообщение № 2',0
5 msg3 DB 'Сообщение № 3',0
6 SECTION .text
7 GLOBAL _start
8 _start
9 jmp _label2
10 _label1
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение № 1'
13 _label2
14 mov eax, msg2 ; Вывод на экран строки
15 call sprintf ; 'Сообщение № 2'
16 jmp _label3
17 _label3
18 mov eax, msg3 ; Вывод на экран строки
19 call sprintf ; 'Сообщение № 3'
20 jmp _end
21 _end
22 call quit ; вызов подпрограммы завершения
23
```

рис.2 Ввод текста программы из листинга 7.1

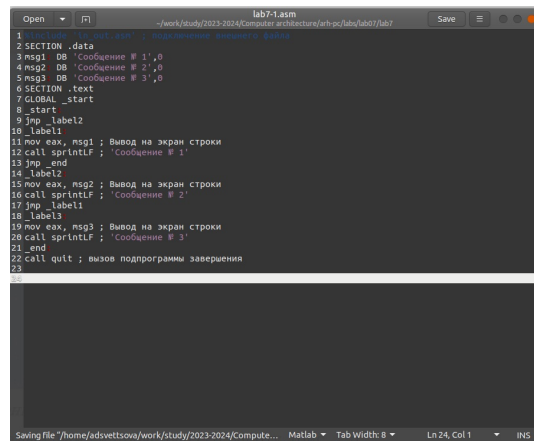
Создаю исполняемый файл и запускаю его. (рис. 3).

```
adsvettsova@bannpc ~/w/s/2/C/arh-pc/l/lab07/lab7 master +1 128 79 $ nasm -f elf lab7-1.asm
adsvettsova@bannpc ~/w/s/2/C/arh-pc/l/lab07/lab7 master +1 128 79 $ ld -m elf_i386 lab7-1.o -o lab7-1
adsvettsova@bannpc ~/w/s/2/C/arh-pc/l/lab07/lab7 master +1 128 79 $ ./lab7-1
Сообщение № 2
Сообщение № 3
adsvettsova@bannpc ~/w/s/2/C/arh-pc/l/lab07/lab7 master +1 128 80 $ 22:35:42
```

рис.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.

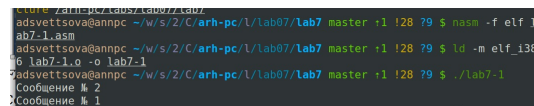
Изменяю программу таким образом, чтобы она выводила сначала 'Сообщение № 2', потом 'Сообщение № 1' и завершала работу. Для этого изменяю текст программы в соответствии с листингом 7.2. (рис. 4).



```
1 2 SECTION .data
3 msg1 DB 'Сообщение # 1',0
4 msg2 DB 'Сообщение # 2',0
5 msg3 DB 'Сообщение # 3',0
6 SECTION .text
7 GLOBAL _start
8 _start
9 jmp _label2
10 _label1
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение # 1'
13 jmp _end
14 _label2
15 mov eax, msg2 ; Вывод на экран строки
16 call sprintf ; 'Сообщение # 2'
17 jmp _label1
18 _label3
19 mov eax, msg3 ; Вывод на экран строки
20 call sprintf ; 'Сообщение # 3'
21 _end
22 call quit ; вызов подпрограммы завершения
23
```

рис.4 Изменение текста программы

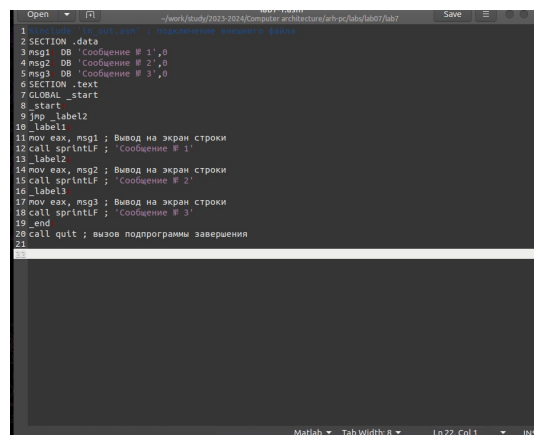
Создаю исполняемый файл и проверяю его работу. (рис. 5).



```
adsvettsova@bannpc ~/w/s/2/C/arrh-pc/1/lab07/lab7 master +1 128 79 $ nasm -f elf 1
lab7-1.asm
adsvettsova@bannpc ~/w/s/2/C/arrh-pc/1/lab07/lab7 master +1 128 79 $ ld -m elf_i386
lab7-1.o -o lab7-1
adsvettsova@bannpc ~/w/s/2/C/arrh-pc/1/lab07/lab7 master +1 128 79 $ ./lab7-1
Сообщение # 2
Сообщение # 1
```

рис.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. 6).



```
1 2 SECTION .data
3 msg1 DB 'Сообщение # 1',0
4 msg2 DB 'Сообщение # 2',0
5 msg3 DB 'Сообщение # 3',0
6 SECTION .text
7 GLOBAL _start
8 _start
9 jmp _label2
10 _label1
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение # 1'
13 jmp _label2
14 mov eax, msg2 ; Вывод на экран строки
15 call sprintf ; 'Сообщение # 2'
16 _label3
17 mov eax, msg3 ; Вывод на экран строки
18 call sprintf ; 'Сообщение # 3'
19 jmp _end
20 _end
21 call quit ; вызов подпрограммы завершения
22
```

рис.6 Изменение текста программы

чтобы вывод программы был следующим: (рис. 7).

```
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab07/lab7 master +1 128 79 $ nasm -f elf l
ab7-1.asm
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab07/lab7 master +1 128 79 $ ld -m elf_i38
6 lab7-1.o -o lab7-1
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab07/lab7 master +1 128 79 $ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab07/lab7 master +1 128 79 $
```

рис.7 Вывод программы

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

Создаю файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. (рис. 8).

```
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab07/lab7 master +1 128 79 $ touch lab7-2.
asm
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab07/lab7 master +1 128 79 $ nasm -f elf l
```

рис.8 Создание файла

Текст программы из листинга 7.3 ввожу в lab7-2.asm. (рис. 9).

```
1 section .data
2 vwordx db "Введите x ",0
3 vwordx db "Введите a ",0
4 vwordx db "Введите b ",0
5 vwordx db "Результат ",0
6 section .bss
7 x resb 80
8 a resb 80
9 section .text
10 global _start
11 _start:
12 mov eax,vwordx
13 call sprint
14 mov ecx,a
15 mov edx,80
16 call sread
17 mov eax,a
18 call atoi
19 cmp ecx,3
20 jl functionx
21 mov eax,vwordx
22 call sprint
23 mov ecx,x
24 mov edx,80
25 call sread
26 mov eax,x
27 call atoi
28 jmp _functiona
29 _functiona:
30 add eax,1
31 jmp _end
32 _functionx:
33 mov ecx,3
34 mul edx
35 jmp _end
36 _end:
37 mov ecx,eax
38 mov eax,vwordx
```

рис.9 Ввод текста программы из листинга 7.3

Создаю исполняемый файл и проверю его работу. (рис. 10).

```
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab07/lab7 master +1 128 79 $ nasm -f elf l
ab7-2.asm
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab07/lab7 master +1 128 79 $ ld -m elf_i38
6 lab7-2.o -o lab7-2
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab07/lab7 master +1 128 79 $ ./lab7-2
Введите B: 10
Наибольшее число: 50
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab07/lab7 master +1 128 79 $ nasm -f elf l
ab7-2.asm
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab07/lab7 master +1 128 79 $ ld -m elf_i38
6 lab7-2.o -o lab7-2
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab07/lab7 master +1 128 79 $ ./lab7-2
Введите B: 100
Наибольшее число: 100
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab07/lab7 master +1 128 79 $
```

рис.10 Проверка работы файла

Файл работает корректно.

4.2 Изучение структуры файлы листинга

Создаю файл листинга для программы из файла lab7-2.asm. (рис. 11).

```
adsvettsova@nannpc ~/w/s/2/C/arb-pc/1/lab07/lab7 master :1 128 79 $ nasm -f elf  
-l lab7-2.lst lab7-2.asm
```

рис.11 Создание файла листинга

Открываю файл листинга lab7-2.lst с помощью текстового редактора и внимательно изучаю его формат и содержимое. (рис. 12).

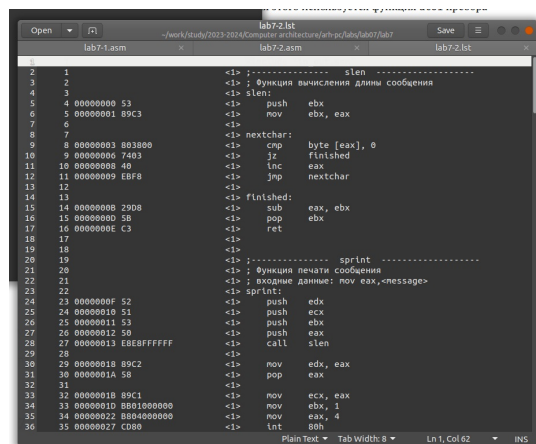


рис.12 Изучение файла листинга

В представленных трех строчках содержатся следующие данные: (рис. 13).

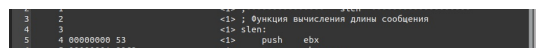


рис.13 Выбранные строки файла

“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. 14).

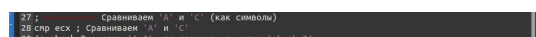


рис.14 Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга. (рис. 15).

```

C:\lab7\2.lst lab7-2.asm
adsvetsova@anncpc ~/w/s/2/C/arb-pc/1/lab07/lab7 master +1 128 710 $ nasm -f elf
-l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands

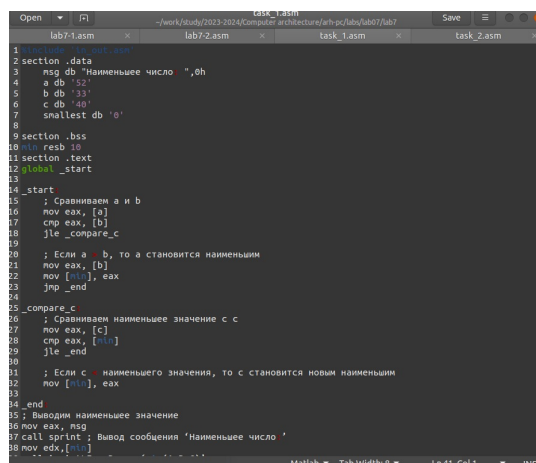
```

рис.15 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.

4.3 Задания для самостоятельной работы

1. 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 № 7. Мой вариант под номером 10, поэтому мои значения - 41, 62 и 35. (рис. 16).



```

1  ;.asm
2 section .data
3 msg db "Наименьшее число: ",0h
4 a db '52'
5 b db '33'
6 c db '40'
7 smallest db '0'
8
9 section .bss
10 min resb 10
11 section .text
12 global _start
13
14 _start:
15     ; Сравниваем a и b
16     mov eax, [a]
17     cmp eax, [b]
18     jle _compare_c
19
20     ; Если a < b, то a становится наименьшим
21     mov eax, [a]
22     mov [min], eax
23     jmp _end
24
25 _compare_c:
26     ; Сравниваем наименьшее значение с c
27     mov eax, [c]
28     cmp eax, [min]
29     jle _end
30
31     ; Если c наименьшего значения, то c становится новым наименьшим
32     mov [min], eax
33
34 _end:
35     ; Выводим наименьшее значение
36     mov eax, msg
37     call sprintf ; Вывод сообщения "Наименьшее число: "
38     mov ebx, [min]

```

рис.16 Написание программы

Создаю исполняемый файл и проверяю его работу, подставляя необходимые значение. (рис. 17).

```

adsvetsova@anncpc ~/w/s/2/C/arb-pc/1/lab07/lab7 master +1 128 79 $ nasm -f elf
task1.asm
adsvetsova@anncpc ~/w/s/2/C/arb-pc/1/lab07/lab7 master +1 128 79 $ ld -m elf_i386 task1.o -o task1
adsvetsova@anncpc ~/w/s/2/C/arb-pc/1/lab07/lab7 master +1 128 79 $ ./task1
Наименьшее число: 33

```

рис.17 Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```

#include 'in_out.asm' section .data msg db "Наименьшее число:",0h
a db '52' b db '33' c db '40' smallest db '0'

```

```

section .bss min resb 10 section .text global _start

```

```

_start: ; Сравниваем a и b mov eax, [a] cmp eax, [b] jle _compare_c

```

; Если $a > b$, то a становится наименьшим

mov eax, [b]

mov [min], eax

jmp _end

_compare_c: ; Сравниваем наименьшее значение с c mov eax, [c]

cmp eax, [min] jle _end

; Если $c <$ наименьшего значения, то c становится новым наименьшим

mov [min], eax

_end: ; Выводим наименьшее значение mov eax, msg call sprint ;

Вывод сообщения 'Наименьшее число:' mov edx,[min] call iprintLF ;

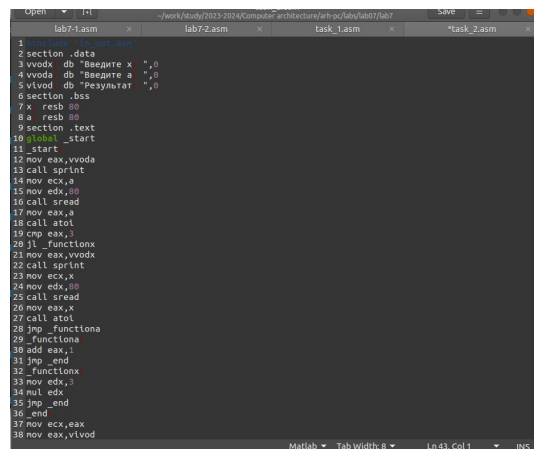
Вывод 'min(A,B,C)' call quit

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

За, при $a < 3$

$x + 1$, при $a \geq 3$

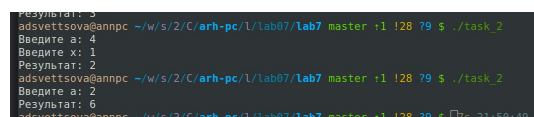
(рис. 18).



```
1 section .data
2     vvdv db "Введите x ",0
3     vvoda db "Введите a ",0
4     vlvod db "Результат ",0
5 section .bss
6     x resb 80
7     a resb 80
8 section .text
9     _start:
10    mov eax, vvdv
11    call sprint
12    mov ecx, a
13    call sread
14    mov edx, 0
15    call atoi
16    cmp ecx, 3
17    jl _functionx
18    mov eax, vlvod
19    call sprint
20    mov ecx, x
21    call sread
22    mov edx, 0
23    call atoi
24    jmp _functiona
25    _functiona:
26    add eax, 1
27    jmp _end
28    _functionx:
29    mov edx, 1
30    mul ecx
31    jmp _end
32    _end:
33    mov ecx, eax
34    call vlvod
```

рис.18 Написание программы

Создаю исполняемый файл и проверяю его работу для значений x и a соответственно: (1, 4), (2, 6). (рис. 19).



```
Результат: 3
advettsova@annpc ~/w/s/2/C/arb-pc/1/lab07/lab7 master +1 128 79 $ ./task_2
Введите x: 1
Введите a: 4
Результат: 2
advettsova@annpc ~/w/s/2/C/arb-pc/1/lab07/lab7 master +1 128 79 $ ./task_2
Введите x: 2
Введите a: 6
Результат: 6
advettsova@annpc ~/w/s/2/C/arb-pc/1/lab07/lab7 master +1 128 79 $
```

рис.19 Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```
%include 'in_out.asm' section .data vvodx: db "Введите x:",0 vvoda:
db "Введите a:",0 vivod: db "Результат:",0 section .bss x: resb 80 a:
resb 80 section .text global _start _start: mov eax,vvoda call sprint
mov ecx,a mov edx,80 call sread mov eax,a call atoi cmp eax,3 jl
_functionx mov eax,vvodx call sprint mov ecx,x mov edx,80 call sread
mov eax,x call atoi jmp _functiona _functiona: add eax,1 jmp _end
_functionx: mov edx,3 mul edx jmp _end _end: mov ecx,eax mov
eax,vivod call sprint mov eax,ecx call iprintLF call quit
```

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).