

Отчет по лабораторной работе №7

Дисциплина: архитектура компьютера

Гомазкова Алина

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 файлы листинга	13
4.3	Задания для самостоятельной работы	14
5	Выводы	17
	Список литературы	18

Список иллюстраций

4.1	Рис. 1 Создание файлов для лабораторной работы	8
4.2	Рис. 2 Ввод текста программы из листинга 7.1	8
4.3	Рис. 3 Запуск программного кода	9
4.4	Рис. 4 Изменение текста программы	9
4.5	Рис. 5 Создание исполняемого файла	10
4.6	Рис. 6 Изменение текста программы	10
4.7	Рис. 7 Вывод программы	11
4.8	Рис. 8 Создание файла	11
4.9	Рис. 9 Ввод текста программы из листинга 7.3	12
4.10	Рис. 10 Проверка работы файла	12
4.11	Рис. 11 Создание файла листинга	13
4.12	Рис. 12 Изучение файла листинга	13
4.13	Рис. 13 Выбранные строки файла	13
4.14	Рис. 14 Удаление выделенного операнда из кода	14
4.15	Рис. 15 Получение файла листинга	14
4.16	Рис. 16 Написание программы	15
4.17	Рис. 17 Запуск файла и проверка его работы	15
4.18	Рис. 18 Выражение для $f(x)$	15
4.19	Рис. 19 Написание программы	16
4.20	Рис. 20 Запуск файла и проверка его работы	16

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 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 `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.

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)

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

Рис. 4.1: Рис. 1 Создание файлов для лабораторной работы

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

```
mc [alinagomazkova@fedora]:~/work/study/2023-2024/Архитектура комп  
lab7-1.asm [-M--] 41 L:[ 1+19 20/ 20] *(649 / 649b) <EOF>  
%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: Рис. 2 Ввод текста программы из листинга 7.1

Создаю исполняемый файл и запускаю его (рис. 3)

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

Рис. 4.3: 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)

```
lab7-1.asm      [----] 41 L:[ 1+21 22/ 22] *(671 / 671b) <EOF>
#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: Рис. 4 Изменение текста программы

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

```

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

```

Рис. 4.5: 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)

```

lab7-1.asm      [-M--] 11 L:[ 1+20 21/ 22] *(608 / 614b) 0010
%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:

```

Рис. 4.6: Рис. 6 Изменение текста программы

Чтобы вывод программы был следующим: (рис. 7)

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

Рис. 4.7: 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)

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

Рис. 4.8: Рис. 8 Создание файла

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

```

lab7-2.asm [-M--] 17 L: [ 1+ 0 1/ 49] *(17 /1743b) 0097
#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 ; 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'

```

Рис. 4.9: Рис. 9 Ввод текста программы из листинга 7.3

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

```

[alinagomazkova@fedora lab07]$ nasm -f elf lab7-2.asm
[alinagomazkova@fedora lab07]$ ld -m elf_i386 lab7-2.o -o lab7-2
[alinagomazkova@fedora lab07]$ ./lab7-2
Введите B: 40
Наибольшее число: 50

```

Рис. 4.10: Рис. 10 Проверка работы файла

Файл работает корректно.

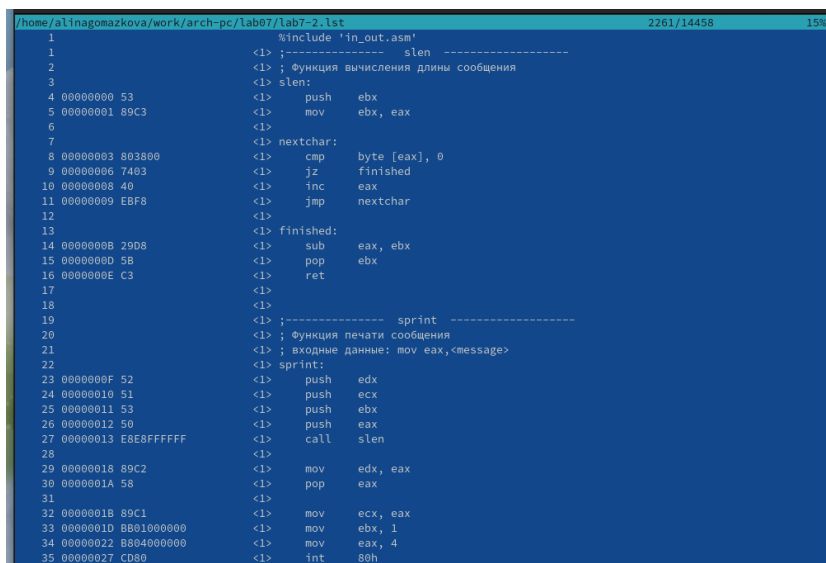
4.2 Изучение структуры файлы листинга

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

```
наибольшее число: 50  
[alinagomazkova@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
```

Рис. 4.11: Рис. 11 Создание файла листинга

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



```
/home/alinagomazkova/work/arch-pc/lab07/lab7-2.lst 2261/14458 15%  
1 %include 'in_out.asm'  
2 <1> ; Функция вычисления длины сообщения  
3 <1> slen:  
4 00000000 53 <1> push ebx  
5 00000001 89C3 <1> mov ebx, eax  
6  
7 <1> nextchar:  
8 00000003 803800 <1> cmp byte [eax], 0  
9 00000006 7403 <1> jz finished  
10 00000008 40 <1> inc eax  
11 00000009 EBF8 <1> jmp nextchar  
12  
13 <1> finished:  
14 0000000B 2908 <1> sub eax, ebx  
15 0000000D 5B <1> pop ebx  
16 0000000E C3 <1> ret  
17  
18  
19 <1> ; Функция печати сообщения  
20 <1> ; входные данные: mov eax, <message>  
21 <1> sprint:  
22 <1> push edx  
23 0000000F 52 <1> push ecx  
24 00000010 51 <1> push ebx  
25 00000011 53 <1> push eax  
26 00000012 50 <1> call slen  
27 00000013 E8E8FFFFFF <1> mov edx, eax  
28 <1> pop eax  
29 00000018 89C2 <1> mov ecx, eax  
30 0000001A 58 <1> mov ebx, 1  
31 <1> mov eax, 4  
32 0000001B 89C1 <1> mov eax, 4  
33 0000001D B801000000 <1> int 80h  
34 00000022 B804000000  
35 00000027 CD80
```

Рис. 4.12: Рис. 12 Изучение файла листинга

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



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

Рис. 4.13: 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. 4.14: Рис. 14 Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга (рис. 15)

Рис. 4.15: 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 9, поэтому мои значения - 24, 98 и 15. (рис. 16)

```

mc [alinagomazkova@10.0.2.15]~/work/arch-pc/lab07
task1.asm [----] 29 L: [ 1+28 29/ 38] *(1064/1299b) 0010 0x00A
%include 'in_out.asm'
section .data
msg db "Наименьшее число: ",0h
A dd '24'
B dd '98'
C dd '15'
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,[C] ; Сравниваем 'A' и 'C'
jg 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'

```

Рис. 4.16: Рис. 16 Написание программы

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

```

[alinagomazkova@10 lab07]$ nasm -f elf task1.asm
[alinagomazkova@10 lab07]$ ld -m elf_i386 -o task1 task1.o
[alinagomazkova@10 lab07]$ ./task1
Наименьшее число: 15
[alinagomazkova@10 lab07]$

```

Рис. 4.17: Рис. 17 Запуск файла и проверка его работы

- Пишу программу, которая для введенных с клавиатуры значений x и a вычисляет значение и выводит результат вычислений заданной для моего варианта функции (рис. 18) (рис. 19)

$$9 \quad \begin{cases} a + x, & x \leq a \\ a, & x > a \end{cases}$$

Рис. 4.18: Рис. 18 Выражение для $f(x)$

```
mc [alinagomazkova@10.0.2.15]:~/work/arch-pc/l
/home/alinagomazkova/work/arch-pc/lab07/task2.asm
%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,vvodx
call sprint
mov ecx,x
mov edx,80
call sread
```

Рис. 4.19: Рис. 19 Написание программы

Создаю исполняемый файл и проверяю его работу для значений x и a соответственно: (5;7) (6;4) (рис. 20)

```
[alinagomazkova@10 lab07]$ nasm -f elf task2.asm
[alinagomazkova@10 lab07]$ ld -m elf_i386 -o task2 task2.o
[alinagomazkova@10 lab07]$ ./task2
Введите x: 5
Введите a: 7
Результат: 7
[alinagomazkova@10 lab07]$ ./task2
Введите x: 6
Введите a: 4
Результат: 4
```

Рис. 4.20: Рис. 20 Запуск файла и проверка его работы

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.

Список литературы

1.Архитектура ЭВМ