

Отчёт по лабораторной работе №5

2023

Просина Ксения Максимовна

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	7
3.1	Реализация переходов в NASM	7
3.2	Изучение структуры файлы листинга	12
3.3	Задание для самостоятельной работы	13
4	Выводы	18
	Список литературы	19

Список иллюстраций

3.1	Создание каталога и файла	7
3.2	Код	8
3.3	Вывод	8
3.4	Проверка	9
3.5	Код	10
3.6	Проверка	11
3.7	Создание	11
3.8	Проверка	11
3.9	Создание	12
3.10	Листинг	13
3.11	Написание кода	14
3.12	Проверка	14
3.13	Проверка	17

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 Теоретическое введение

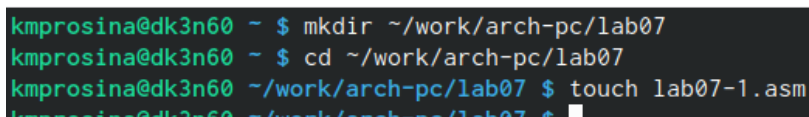
Безусловный переход выполняется инструкцией `jmp` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление: `jmp` Адрес перехода может быть либо меткой, либо адресом области памяти, в которую предварительно помещен указатель перехода. Кроме того, в качестве операнда можно использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре Таблица 7.1. Типы операндов инструкции `jmp`

Тип операнда	Описание
<code>jmp label</code>	переход на метку <code>label</code>
<code>jmp [label]</code>	переход по адресу в памяти, помеченному меткой <code>label</code>
<code>jmp eax</code>	переход по адресу из регистра <code>eax</code>

3 Выполнение лабораторной работы

3.1 Реализация переходов в NASM

1. Создайте каталог для программ лабораторной работы No 7, перейдите в него и создайте файл lab7-1.asm:



```
kmprosina@dk3n60 ~ $ mkdir ~/work/arch-pc/lab07
kmprosina@dk3n60 ~ $ cd ~/work/arch-pc/lab07
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ touch lab07-1.asm
kmprosina@dk3n60 ~/work/arch-pc/lab07 $
```

Рис. 3.1: Создание каталога и файла

2. Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`. Введите в файл `lab7-1.asm` текст программы из листинга

```

1 %include 'in_out.asm'
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
12 call sprintLF
13 _label2:
14 mov eax, msg2
15 call sprintLF
16 _label3:
17 mov eax, msg3
18 call sprintLF
19 _end:
20 call quit

```

Рис. 3.2: Код

Создайте исполняемый файл и запустите его. Результат работы данной программы будет следующим:

```

nasm: fatal: unable to open input file 'meow.asm': no such file or directory
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ nasm -f elf lab07-1.asm
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab07-1 lab07-1.o
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ ./lab07-1
Сообщение № 2
Сообщение № 3

```

Рис. 3.3: Вывод

Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`,

пропустив вывод первого сообщения. Инструкция `jmp` позволяет осуществлять переходы не только вперед но и назад. Изменим программу таким образом, чтобы она выводила сначала ‘Сообщение No 2’, потом ‘Сообщение No 1’ и завершала работу. Для этого в текст программы после вывода сообщения No 2 добавим инструкцию `jmp` с меткой `_label1` (т.е. переход к инструкциям вывода сообщения No 1) и после вывода сообщения No 1 добавим инструкцию `jmp` с меткой `_end` (т.е. переход к инструкции `call quit`). Измените текст программы в соответствии с листингом 7.2. Создайте исполняемый файл и проверьте его работу.

```
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ nasm -f elf lab07-1.asm
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab07-1 lab07-1.o
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ ./lab07-1
Сообщение № 2
Сообщение № 1
```

Рис. 3.4: Проверка

Измените текст программы добавив или изменив инструкции `jmp`, чтобы вывод программы был следующим:

```
1 %include 'in_out.asm'
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
10 jmp _label3
11
12 _label1:
13 mov eax, msg1
14 call sprintf
15 jmp _end
16
17 _label2:
18 mov eax, msg2
19 call sprintf
20 jmp _label1
21
22 _label3:
23 mov eax, msg3
24 call sprintf
25 jmp _label2
26
27 _end:
28 call quit
```

Рис. 3.5: Код

```

Сообщение № 1
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ nasm -f elf lab07-1.asm
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab07-1 lab07-1.o
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ ./lab07-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
kmprosina@dk3n60 ~/work/arch-pc/lab07 $

```

Рис. 3.6: Проверка

- Использование инструкции `jmp` приводит к переходу в любом случае. Однако, часто при написании программ необходимо использовать условные переходы, т.е. переход должен происходить если выполнено какое-либо условие. В качестве примера 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. Создайте файл `lab7-2.asm` в каталоге `~/work/arch-pc/lab07`. Внимательно изучите текст программы из листинга 7.3 и введите в `lab7-2.asm`. Создайте исполняемый файл и проверьте его работу для разных значений В.

```

Сообщение № 1
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ touch lab07-2.asm
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ ls
in_out.asm lab07-1 lab07-1.asm lab07-1.o lab07-2.asm
kmprosina@dk3n60 ~/work/arch-pc/lab07 $

```

Рис. 3.7: Создание

```

kmprosina@dk3n60 ~/work/arch-pc/lab07 $ gedit lab07-2.asm
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ nasm -f elf lab07-2.asm
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab07-2 lab07-2.o
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ ./lab07-2
Введите В: 14
Наибольшее число: 50
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ ./lab07-2
Введите В: 65
Наибольшее число: 65
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ ./lab07-2
Введите В: 1
Наибольшее число: 50
kmprosina@dk3n60 ~/work/arch-pc/lab07 $ ./lab07-2

```

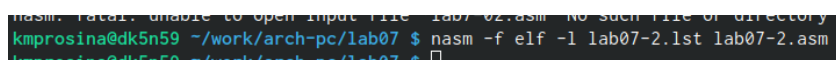
Рис. 3.8: Проверка

Обратите внимание, в данном примере переменные А и С сравниваются как символы, а переменная В и максимум из А и С как числа (для этого используется

функция `atoi` преобразования символа в число). Это сделано для демонстрации того, как сравниваются данные. Данную программу можно упростить и сравнивать все 3 переменные как символы (т.е. не использовать функцию `atoi`). Однако если переменные преобразовать из символов числа, над ними можно корректно проводить арифметические операции.

3.2 Изучение структуры файлы листинга

4. Обычно `nasm` создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ `-l` и задав имя файла листинга в командной строке. Создайте файл листинга для программы из файла `lab7-2.asm` `nasm -f elf -l lab7-2.lst lab7-2.asm`



```
nasm: fatal: unable to open input file 'lab7-02.asm': No such file or directory
kmprosina@dk5n59 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
kmprosina@dk5n59 ~/work/arch-pc/lab07 $
```

Рис. 3.9: Создание

Откройте файл листинга `lab7-2.lst` с помощью любого текстового редактора, например `mcedit`: `mcedit lab7-2.lst`

```

1 1 %include 'in_out.asm'
2 1 <1> ;----- slen -----
3 2 <1> ; Функция вычисления длины сообщения
4 3 <1> slen:
5 4 00000000 53 <1> push ebx
6 5 00000001 89C3 <1> mov ebx, eax
7 6 <1>
8 7 <1> nextchar:
9 8 00000003 803800 <1> cmp byte [eax], 0
10 9 00000006 7403 <1> jz finished
11 10 00000008 40 <1> inc eax
12 11 00000009 EBF8 <1> jmp nextchar
13 12 <1>
14 13 <1> finished:
15 14 0000000B 29D8 <1> sub eax, ebx
16 15 0000000D 5B <1> pop ebx
17 16 0000000E C3 <1> ret
18 17 <1>
19 18 <1>
20 19 <1> ;----- sprint -----
21 20 <1> ; Функция печати сообщения
22 21 <1> ; входные данные: mov eax, <message>
23 22 <1> sprint:
24 23 0000000F 52 <1> push edx
25 24 00000010 51 <1> push ecx
26 25 00000011 53 <1> push ebx
27 26 00000012 50 <1> push eax
28 27 00000013 E8E8FFFFFF <1> call slen
29 28 <1>
30 29 00000018 89C2 <1> mov edx, eax
31 30 0000001A 58 <1> pop eax
32 31 <1>
33 32 0000001B 89C1 <1> mov ecx, eax
34 33 0000001D BB01000000 <1> mov ebx, 1
35 34 00000022 B804000000 <1> mov eax, 4
36 35 00000027 CD80 <1> int 80h
37 36 <1>
38 37 00000029 5B <1> pop ebx
39 38 0000002A 59 <1> pop ecx
40 39 0000002B 5A <1> pop edx
41 40 0000002C C3 <1> ret
42 41 <1>
43 42 <1>
44 43 <1> ;----- sprintf -----
45 44 <1> ; Функция печати сообщения с переводом строки

```

Текст Ширина таблицы: 8 Стр 1, Стлб 1 ВСТ

Рис. 3.10: Листинг

3.3 Задание для самостоятельной работы

1. Напишите программу нахождения наименьшей из 3 целочисленных переменных \boxed{x} , \boxed{y} и \boxed{z} . Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы No 7. Создайте исполняемый файл и проверьте его работу.

```

1 %include 'in_out.asm'
2 section .data
3 msg1 db "Наименьшее число: ",0h
4 A dd '46'
5 B dd '32'
6 C dd '74'
7
8 section .bss
9 min resb 10
10
11 section .text
12 global _start
13 _start:
14
15 mov eax,B
16 call atoi
17 mov [B],eax
18
19 mov ecx,[A]
20 mov [min],ecx
21
22 cmp ecx,[C]
23 jl check_B
24 mov ecx,[C]
25 mov [min],ecx
26
27 check_B:
28 mov eax,min
29 call atoi
30 mov [min],eax
31
32 mov ecx,[min]
33 cmp ecx,[B]
34 jl fin
35 mov ecx,[B]
36 mov [min],ecx
37
38 fin:
39 mov eax,msg1
40 call sprint
41 mov eax,[min]
42 call iprintLF
43 call quit

```

Рис. 3.11: Написание кода

Для выполнения задания нужно было создать 3 числа, вписать их в код и сравнить их между собой. В отличие от прошлого кода, для этого я использовала не JG, а JL, что значит, что там сравниваются числа и записывается наименьшее

```

kmprosina@dk5n59 ~/work/arch-pc/lab07 $ ./lab07-3
Наименьшее число: 32

```

Рис. 3.12: Проверка

2. Напишите программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции $f(x, a)$ и выводит результат вычислений. Вид функции $f(x, a)$ выбрать из таблицы 7.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы No 7. Создайте исполняемый файл и проверьте его работу для значений x и a из 7.6.

Получившийся код:

```
%include 'in_out.asm'

section .data
msg1 db 'Введите x:',0h
msg2 db 'Введите a:',0h
msg3 db "x + a =",0h
msg4 db "x =",0h
section .bss
B resb 10 ; x
C resb 10 ; a
A resb 10 ; ответ 1
D resb 10 ; ответ 2

section .text
global _start
_start:

; ----- Вывод сообщения 'Введите x:' mov eax,msg1
call sprint

; ----- Ввод 'x'
mov ecx,B
mov edx,10
call sread
```

```

; ——— Преобразование 'х' из символа в число
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'х'

; ——— Вывод сообщения 'Введите а:'
mov eax,msg2
call sprint
; ——— Ввод 'а'
mov ecx,C
mov edx,10
call sread
; ——— Преобразование 'а' из символа в число
mov eax,C
call atoi ; Вызов подпрограммы перевода символа в число
mov [C],eax ; запись преобразованного числа в 'В'

; ——— Преобразование 'а' из символа в число
mov eax,A
call atoi ; Вызов подпрограммы перевода символа в число
mov [A],eax ; запись преобразованного числа в 'В'
; ——— Преобразование 'а' из символа в число
mov eax,D
call atoi ; Вызов подпрограммы перевода символа в число
mov [D],eax ; запись преобразованного числа в 'В'

mov eax,[B] ; запись х
add eax,[C] ; добавление к х а
mov [A],eax ; запись в ответ 1 получившуюся сумму

```



```
mov edx,[B] ; запись x
mov [D],edx ; x в ответ 2
```

; ——— Вывод результата

fin:

```
mov eax, msg3
```

```
call sprint
```

```
mov eax,[A]
```

```
call iprintLF
```

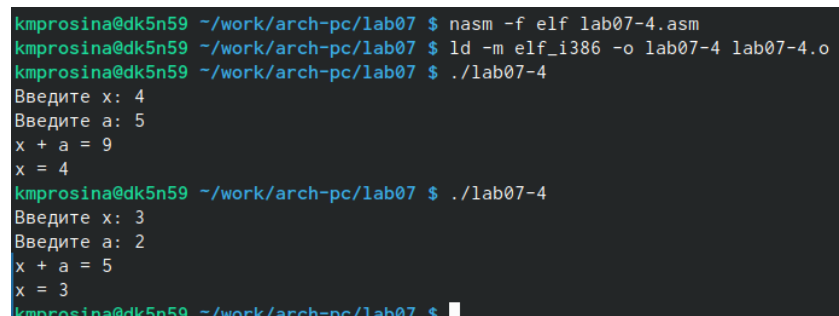
```
mov eax, msg4
```

```
call sprint
```

```
mov eax,[D]
```

```
call iprintLF
```

```
call quit
```



```
kmprosina@dk5n59 ~/work/arch-pc/lab07 $ nasm -f elf lab07-4.asm
kmprosina@dk5n59 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab07-4 lab07-4.o
kmprosina@dk5n59 ~/work/arch-pc/lab07 $ ./lab07-4
Введите x: 4
Введите a: 5
x + a = 9
x = 4
kmprosina@dk5n59 ~/work/arch-pc/lab07 $ ./lab07-4
Введите x: 3
Введите a: 2
x + a = 5
x = 3
kmprosina@dk5n59 ~/work/arch-pc/lab07 $
```

Рис. 3.13: Проверка

4 Выводы

Изучили команды условного и безусловного переходов. Приобрели навыки написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

Список литературы

https://esystem.rudn.ru/pluginfile.php/2089665/mod_resource/content/0/%D0%9B%D0%B0%D0%