

Отчет по лабораторной работе №5

Архитектура компьютера

Никифоров Захар Сергеевич

Содержание

1	Цель работы	5
2	Порядок выполнения работы	6
2.1	Реализация переходов в NASM	6
2.2	Изучение структуры файлы листинга	11
3	Задание для самостоятельной работы	13
4	Выводы	17

Список иллюстраций

2.1	Скриншот терминала тс 1	6
2.2	Скриншот терминала тс 2	7
2.3	Скриншот терминала тс 3	7
2.4	Скриншот терминала тс 4	8
2.5	Скриншот терминала тс 5	8
2.6	Скриншот терминала тс 6	9
2.7	Скриншот терминала тс 7	9
2.8	Скриншот терминала тс 8	10
2.9	Скриншот терминала тс 9	10
2.10	Скриншот терминала тс 10	11
2.11	Скриншот терминала тс 11	11
2.12	Скриншот терминала тс 12	11
2.13	Скриншот терминала тс 13	11
2.14	Скриншот терминала тс 14	12
3.1	Скриншот терминала тс 15	13
3.2	Скриншот терминала тс 16	14
3.3	Скриншот терминала тс 17	14
3.4	Скриншот терминала тс 18	15
3.5	Скриншот терминала тс 19	15
3.6	Скриншот терминала тс 20	16
3.7	Скриншот терминала тс 21	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 Порядок выполнения работы

2.1 Реализация переходов в NASM

Создаем каталог *lab07*, а в нем создаем файл *lab07.asm*, записываем код из *Листинг 7.1* и компилируем его.

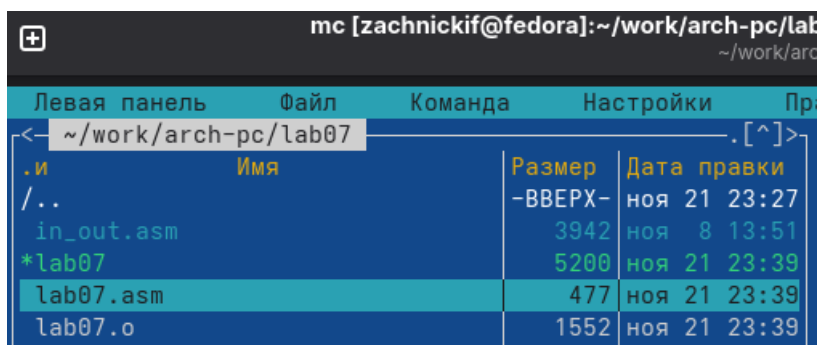


Рисунок 2.1: Скриншот терминала mc 1

```

lab07.asm [----] 21 L:[ 1+30 31/ 31] *(477 / 477b) <EOF>
#include 'in_out.asm'

SECTION .data
msg1: DB 'Сообщение №1', 0
msg2: DB 'Сообщение №2', 0
msg3: DB 'Сообщение №3', 0
buf: RESB 80
...
SECTION .text
GLOBAL _start
<----->_start:
<----->
<----->jmp _label2
<----->
<----->_label1:
<----->    mov eax, msg1
<----->    call sprintf
<----->
<----->_label2:
<----->    mov eax, msg2
<----->    call sprintf
<----->
<----->_label3:
<----->    mov eax, msg3
<----->    call sprintf
<----->...
<----->_end:
<----->    mov ecx, buf
<----->    mov edx, 80
<----->    call sread
<----->    call quit

```

Рисунок 2.2: Скриншот терминала тс 2

```

zachnickif@fedora:~/work/arch-pc/lab07$ ./lab07
Сообщение №2
Сообщение №3

```

Рисунок 2.3: Скриншот терминала тс 3

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

Далее изменим код программы согласно листингу 7.2 и заново соберем файл.

```

lab07.asm      [----] 23 L:[ 1+21 22/ 31] *(377 / 504b) 0010 0x00A
#include 'in_out.asm'

SECTION .data
msg1: DB 'Сообщение №1', 0
msg2: DB 'Сообщение №2', 0
msg3: DB 'Сообщение №3', 0
buf: RESB 80
....
SECTION .text
GLOBAL _start
<----->_start:
<----->
<----->jmp _label2
<----->
<----->_label1:
<----->    mov eax, msg1
<----->    call sprintf
<----->    jmp _end
<----->_label2:
<----->    mov eax, msg2
<----->    call sprintf
<----->    jmp _label1
<----->_label3:
<----->    mov eax, msg3
<----->    call sprintf
<----->....
<----->_end:
<----->    mov ecx, buf
<----->    mov edx, 80
<----->    call sread
<----->    call quit

```

Рисунок 2.4: Скриншот терминала тс 4

```

zachnickif@fedora:~/work/arch-pc/lab07$ ./lab07
Сообщение №2
Сообщение №1

```

Рисунок 2.5: Скриншот терминала тс 5

На выход мы получаем вывод сначала второго, а потом первого сообщения.
Теперь изменим код, чтобы выводило сообщения с 3-го по 1-е.


```

lab07.asm      [----] 20 L:[ 1+17 18/ 32] *(312 / 521b) 0010 0x00A
#include 'in_out.asm'

SECTION .data
msg1: DB 'Сообщение №1', 0
msg2: DB 'Сообщение №2', 0
msg3: DB 'Сообщение №3', 0
buf: RESB 80
....
SECTION .text
GLOBAL _start
<----->_start:
<----->
<----->jmp _label3
<----->
<----->_label1:
<----->    mov eax, msg1
<----->    call sprintLF
<----->    jmp _end
<----->_label2:
<----->    mov eax, msg2
<----->    call sprintLF
<----->    jmp _label1
<----->_label3:
<----->    mov eax, msg3
<----->    call sprintLF
<----->    jmp _label2
<----->....
<----->_end:
<----->    mov ecx, buf
<----->    mov edx, 80
<----->    call sread
<----->    call quit

```

Рисунок 2.6: Скриншот терминала тс 6

```

zachnickif@fedora:~/work/arch-pc/lab07$ ./lab07
Сообщение №3
Сообщение №2
Сообщение №1

```

Рисунок 2.7: Скриншот терминала тс 7

Задание выполнено успешно.

Теперь создадим файл *lab7-2.asm* и запишем в него код из листинга 7.3, а потом соберем файл

Левая панель	Файл	Команда	Настройки	Правая
<-	~/work/arch-pc/lab07		.	>[-]
.и	Имя	Размер	Дата правки	.и
/..		-ВВЕРХ-	ноя 21 23:27	/..
in_out.asm		3942	ноя 8 13:51	ar
*lab07		5200	ноя 22 00:40	ar
lab07.asm		521	ноя 22 00:39	ar
lab07.o		1568	ноя 22 00:39	ar
*lab7-2		5244	ноя 22 00:46	in
lab7-2.asm		796	ноя 22 00:45	*la
lab7-2.o		1808	ноя 22 00:45	la

Рисунок 2.8: Скриншот терминала тс 8

```

/home/vernel/work/arch-pc/lab07/lab7-2.asm
#include 'in_out.asm'

SECTION .data
    msg1 db 'Введите B: ', 0h
    msg2 db 'Наибольшее число: ', 0h
    A dd '20'
    C dd '50'
    buf: RESB 80

SECTION .bss
    max resb 10
    B resb 10

SECTION .text
    GLOBAL _start
_start:
    mov eax, msg1
    call sprint
    mov ecx, B
    mov edx, 10
    call sread

    mov eax, B
    call atoi

    mov [B], eax
    mov ecx, [A]
    mov [max], ecx
    cmp ecx, [C]
    jg check_B
    mov ecx, [C]
    mov [max], ecx

check_B:
    mov eax, max
    call atoi

    mov [max], eax

```

Рисунок 2.9: Скриншот терминала тс 9

```

zachnickif@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 225
Наибольшее число: 225

```

Рисунок 2.10: Скриншот терминала тс 10

```

zachnickif@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 10
Наибольшее число: 50

```

Рисунок 2.11: Скриншот терминала тс 11

Программа работает корректно и исправно находит наибольшее.

2.2 Изучение структуры файлы листинга

Создадим файл листинга из *lab7-2.asm* с помощью команды *nasm -f elf -l lab7-2.lst lab7-2.asm*

*lab7-2	5244	ноя 22 00:46
lab7-2.asm	796	ноя 22 00:45
lab7-2.lst	13839	ноя 22 00:53
lab7-2.o	1808	ноя 22 00:53

Рисунок 2.12: Скриншот терминала тс 12

Теперь выберем три строки, а именно с 49-й по 51-ю, и объясним их.

49 00000032 50	<1>	push	eax
50 00000033 B80A000000	<1>	mov	eax, 0AH
51 00000038 50	<1>	push	eax

Рисунок 2.13: Скриншот терминала тс 13

В начале каждой строки идёт ее номер, а потом адрес. Например, адрес строки 49 это *00000032*, т.е адрес инструкции *push eax* в сегменте *.text*. Она кодируется одним байтом, поэтому за ней идет адрес *00000033*, но потом идет инструкция *mov eax*,

которая занимает уже 5 байт, поэтому адрес следующего *push eax* уже 00000038. После адреса идут наши инструкции в виде машинного кода. Сами же команды значат то, что им нужно сделать. Например, *push eax* кладет текущее значение регистра *eax* в стек, после нее *mov eax* загружает значение 0Ah, которое является символом перевода строки, в конце еще одна *push eax* кладет в стек загруженное значение 0Ah.

Теперь в коде найдем инструкцию с двумя операндами, например, *mov ecx, B* и уберем *B*.

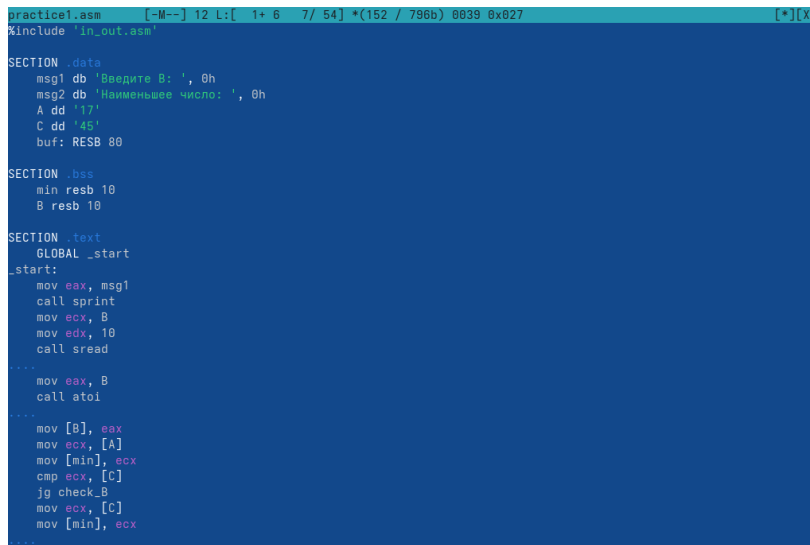
```
14                                SECTION .text
15                                GLOBAL _start
16                                _start:
17 000000E0 B8[00000000]          mov eax, msg1
18 000000ED E81DFFFFFF          call sprint
19                                mov ecx
19 ***** error: invalid combination of opcode and operands
```

Рисунок 2.14: Скриншот терминала mc 14

В таком виде эта инструкция бессмысленна, так как операнда обязательно должно быть два. Листинг создается, но появляется строка об ошибке.

3 Задание для самостоятельной работы

1. Создаем файл *practice1.asm*. Используя в качестве примера код из листинга 7.2, видоизменяем его и немного меняем логику, а также избавляемся от необходимости перевода из строки в число, записав А и С сразу в качестве числа. Собираем файл и проверяем работу с данными из варианта 1.



```
practice1.asm [-M--] 12 L:[ 1+ 6 7/ 54] *(152 / 796b) 0039 0x027 [*][X]
%include 'in_out.asm'

SECTION .data
msg1 db 'Введите B: ', 0h
msg2 db 'Наименьшее число: ', 0h
A dd '17'
C dd '45'
buf: RESB 80

SECTION .bss
min resb 10
B resb 10

SECTION .text
GLOBAL _start
_start:
    mov eax, msg1
    call sprint
    mov ecx, B
    mov edx, 10
    call sread
    ...
    mov eax, B
    call atoi
    ...
    mov [B], eax
    mov ecx, [A]
    mov [min], ecx
    cmp ecx, [C]
    jg check_B
    mov ecx, [C]
    mov [min], ecx
```

Рисунок 3.1: Скриншот терминала mc 15

```
/home/vernel/work/arch-pc/lab07/practice1.asm
#include 'in_out.asm'
```

```
SECTION .data
    msg1 db 'Введите B: ', 0h
    msg2 db 'Наименьшее число: ', 0h
    msg3 db 'Значение A: ', 0h
    msg4 db 'Значение C: ', 0h
    A dd 17
    C dd 45
```

```
SECTION .bss
    min resb 10
    B resb 10
    buf: RESB 80
```

```
SECTION .text
    GLOBAL _start
```

```
_start:
    mov eax, msg3
    call sprint
    mov eax, [A]
    call iprintLF
```

```
    mov eax, msg4
    call sprint
    mov eax, [C]
    call iprintLF
```

```
    mov eax, msg1
    call sprint
    mov ecx, B
    mov edx, 10
    call sread
```

```
    mov eax, B
    call atoi
```

```
    mov [B], eax
```

Рисунок 3.2: Скриншот терминала тс 16

```
    mov [min], ecx
    cmp ecx, [C]
    jl check_B
    mov ecx, [C]
    mov [min], ecx
```

```
check_B:
    mov ecx, [min]
    cmp ecx, [B]
    jl fin
    mov ecx, [B]
    mov [min], ecx
```

```
fin:
    mov eax, msg2
    call sprint

    mov eax, [min]
    call iprintLF
```

```
    mov ecx, buf
    mov edx, 80
    call sread
    call quit
```

Рисунок 3.3: Скриншот терминала тс 17

```
zachnickif@fedora:~/work/arch-pc/lab07$ ./practice1
Значение A: 17
Значение C: 45
Введите B: 23
Наименьшее число: 17
```

Рисунок 3.4: Скриншот терминала тс 18

Программа работает корректно, задание успешно выполнено.

2. Создаем файл *practice2.asm*. Пишем в нем код, который будет запрашивать два значения, а потом согласно заданной функции из варианта 1 выдавать ответ. Соберем файл и проверим корректность работы используя данные из варианта 1.

```
/home/vernel/work/arch-pc/lab07/practice2.asm 60
#include 'in_out.asm'

SECTION .data
    msg1 db 'Введите x: ', 0h
    msg2 db 'Введите a: ', 0h
    msg3 db 'Значение f(x): ', 0h
SECTION .bss
    x resb 10
    a resb 10
    res resb 4
    buf resb 80
SECTION .text
    GLOBAL _start
    _start:
        mov eax, msg1
        call sprint

        mov ecx, x
        mov edx, 10
        call sread

        mov eax, x
        call atoi
        mov [x], eax

        mov eax, msg2
        call sprint

        mov ecx, a
        mov edx, 10
        call sread

        mov eax, a
        call atoi
        mov [a], eax

        mov eax, [x]
        mov ebx, [a]
```

Рисунок 3.5: Скриншот терминала тс 19

```

mov ebx, [a]
cmp eax, ebx
jl less_case
mov dword [res], 8
jmp print_result

less_case:
mov eax, [a]
add eax, [a]
sub eax, [x]
mov [res], eax

print_result:
mov eax, msg3
call sprintf

mov eax, [res]
call iprintLF

mov ecx, buf
mov edx, 80
call sread
call quit

```

Рисунок 3.6: Скриншот терминала mc 20

```

zachnickif@fedora:~/work/arch-pc/lab07$ ./practice2
Введите x: 1
Введите a: 2
Значение f(x): 3

zachnickif@fedora:~/work/arch-pc/lab07$ ./practice2
Введите x: 2
Введите a: 1
Значение f(x): 8

```

Рисунок 3.7: Скриншот терминала mc 21

Программа работает корректно, проверив аналитически, я пришел к выводу, что и арифметически корректно тоже.

4 Выводы

В ходе работы были изучены команды условного и безусловного переходов, а также приобретены навыки написания программ с использованием переходов. Знакомство с назначением и структурой файлов листинга прошло успешно.