

Отчет по лабораторной работе №7

Иванова Анастасия Сергеевна

Содержание

1 Цель работы	5
2 Выполнение лабораторной работы	6
3 Задание для самостоятельной работы	13
4 Вывод	17
Список литературы	18

Список иллюстраций

2.1	Создание каталога и файла	6
2.2	Ввод программы	7
2.3	Проверка работы программы	7
2.4	Изменение программы	8
2.5	Проверка работы программы	9
2.6	Изменение программы	9
2.7	Изменение программы	9
2.8	Проверка работы программы	10
2.9	Ввод программы	11
2.10	Проверка работы программы	11
2.11	Просмотр содержимого файла	12
3.1	Выполнение задания	13
3.2	Выполнение задания	14
3.3	Выполнение задания	15
3.4	Выполнение задания	15
3.5	Выполнение задания	15
3.6	Выполнение задания	16

Список таблиц

1 Цель работы

Изучить команды условного и безусловного перехода. Приобрести навыки написания программ с использованием переходов. Познакомство с назначением и структурой файла листинга.

2 Выполнение лабораторной работы

1. Создадим каталог для программам лабораторной работы № 7, перейдем в него и создадим файл lab7-1.asm рис. 2.1:

```
уже есть.  
asinova1:~$cd ~/work/arch-pc/lab07  
asinova1:~/work/arch-pc/lab07$touch lab7-1.asm  
asinova1:~/work/arch-pc/lab07$
```

Рисунок 2.1: Создание каталога и файла

2. Инструкция jmp в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции jmp. Введем в файл lab7-1.asm текст программы рис. 2.2:

```
%include 'in_out.asm'

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
call sprintLF
jmp _end

_label2:
mov eax, msg2
call sprintLF
jmp _label1

_label3:
mov eax, msg3
call sprintLF

_end:
call quit
```

Рисунок 2.2: Ввод программы

Создайте исполняемый файл и запустите его рис. 2.3:

```
asinova1:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
asinova1:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
asinova1:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
```

Рисунок 2.3: Проверка работы программы

Таким образом, использование инструкции jmp _label2 меняет порядок исполнения программы.

нения 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 № 2“, потом „Сообщение № 1“ и завершала работу. Изменим текст программы рис. 2.4:

```
%include 'in_out.asm'

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
call sprintLF
jmp _end

_label2:
mov eax, msg2
call sprintLF
jmp _label1

_label3:
mov eax, msg3
call sprintLF

_end:
call quit
```

Рисунок 2.4: Изменение программы

Создадим исполняемый файл и проверим его работу рис. 2.5:

```
asinova1:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
asinova1:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
asinova1:~/work/arch-pc/lab07$/lab7-1
Сообщение № 2
Сообщение № 1
asinova1:~/work/arch-pc/lab07$
```

Рисунок 2.5: Проверка работы программы

Изменим текст программы, добавив jmp, чтобы вывод программы был следующим: user@dk4n31:~\$./lab7-1 Сообщение № 3 Сообщение № 2 Сообщение № 1 user@dk4n31:~\$

рис. 2.6:

```
%include 'in_out.asm'
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
call sprintLF
_label2:
mov eax, msg2
call sprintLF
_label3:
mov eax, msg3
call sprintLF
_end:
call quit
```

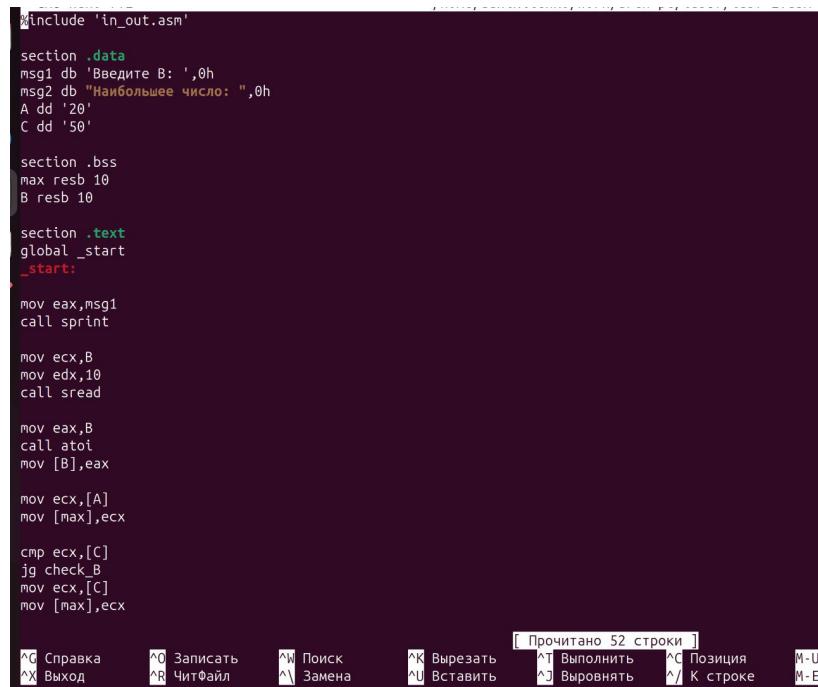
Рисунок 2.6: Изменение программы

рис. 2.7:

```
as inova1:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
as inova1:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
as inova1:~/work/arch-pc/lab07$/lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
as inova1:~/work/arch-pc/lab07$
```

Рисунок 2.7: Изменение программы

рис. 2.8:



```
#include 'in_out.asm'

section .data
msg1 db 'Введите В: ',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:

    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

    cmp max,[B]
    jg check_C
    mov max,[B]
    mov [max],ecx

    cmp max,[C]
    jg check_C
    mov max,[C]
    mov [max],ecx

    mov eax,max
    call sprint

    mov eax,0
    call quit

check_B:
    mov eax,[B]
    mov [max],eax
    jmp check_C

check_C:
    mov eax,[C]
    mov [max],eax
```

Рисунок 2.8: Проверка работы программы

3. 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: A, B и C. Значения для A и C задаются в программе, значение B вводиться с клавиатуры. Создадим файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. Внимательно изучим текст программы и введем в lab7-2.asm рис. 2.11

```

/home/antikitenko/work/arch-pc/lab07/Lab7-2.lst [-*-] 0 L:[ 1+ 0 1/228] *(0 /13388D) 0032 0x020
1           %include 'in_out.asm'
2           ;-----slen -----
3           ; Функция вычисления длины сообщения
4           slen:
5           push    ebx
6           mov     ebx, eax
7           ..
8           nextchar:
9           cmp     byte [eax], 0
10          jz      finished
11          inc     eax
12          jmp     nextchar
13          .
14          finished:
15          sub    eax, ebx
16          pop    ebx
17          ret
18          .
19          ;-----sprint -----
20          ; Функция печати сообщения
21          ; входные данные: mov eax,<message>
22          sprint:
23          push   edx
24          push   ecx
25          push   ebx
26          push   eax
27          call   slen
28          .
29          mov    edx, eax
30          pop    eax
31          .
32          mov    ecx, eax
33          mov    ebx, 1
34          mov    eax, 4
35          int    80h
36          .

```

Рисунок 2.9: Ввод программы

Создадим исполняемый файл и проверим его работу для разных значений В рис. 2.10:

```

aslinova1:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
aslinova1:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
aslinova1:~/work/arch-pc/lab07$ ./lab7-2
.lab7-2: команда не найдена
aslinova1:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 25
Наибольшее число: 50
aslinova1:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 60
Наибольшее число: 60
aslinova1:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 10
Наибольшее число: 50
aslinova1:~/work/arch-pc/lab07$ 

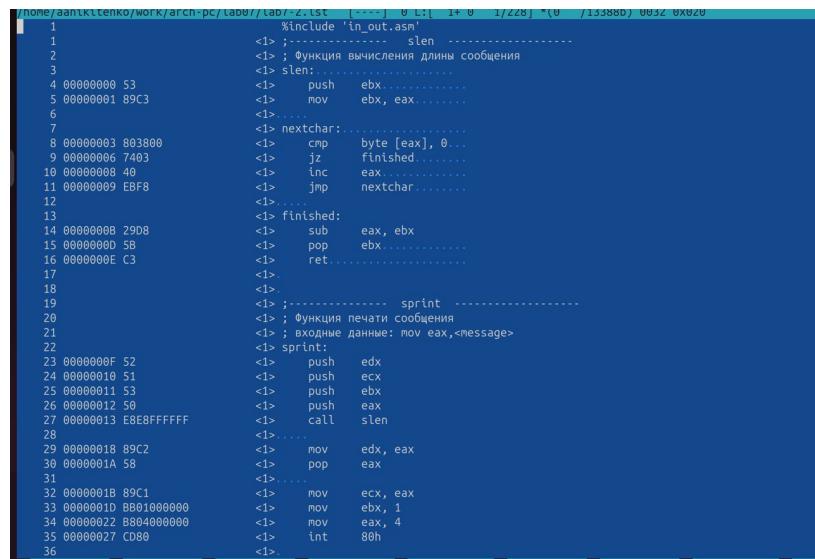
```

Рисунок 2.10: Проверка работы программы

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.

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. Откроем файл с программой lab7-2.asm и в инструкции

с двумя operandами удалим один operand. Выполним трансляцию с получением файла листинга рис. 2.11:



```
/home/dantkktenko/work/arch-pc/lab07/lab-2.lst [---] 0 L[ 1+ 0 1/228] *(0 /13588b) 0032 0x020
1          %include 'in_out.asm'
2          ; Функция вычисления длины сообщения
3          .text
4 00000000 53    push    ebx
5 00000001 89C3  mov     ebx, eax
6          .....
7          .nextchar:
8 00000003 803800  cmp     byte [eax], 0
9 00000006 7403  jz      finished
10 00000008 40   inc     eax
11 00000009 EBF8  jmp     nextchar
12          .....
13          .finished:
14 0000000B 2908  sub     eax, ebx
15 0000000D 5B   pop     ebx
16 0000000E C3   ret
17          .....
18          .....
19          .sprint:
20          ; Функция печати сообщения
21          ; входные данные: mov eax,<message>
22          .sprint:
23 0000000F 52   push    edx
24 00000010 51   push    ecx
25 00000011 53   push    ebx
26 00000012 50   push    eax
27 00000013 EBEBFFFF  call    strlen
28          .....
29 00000018 89C2  mov     edx, eax
30 0000001A 58   pop     eax
31          .....
32 0000001B 89C1  mov     ecx, eax
33 0000001D BB01000000  mov     ebx, 1
34 00000022 B804000000  mov     eax, 4
35 00000027 CD80  int     80h
36          .....
```

Рисунок 2.11: Просмотр содержимого файла

Объяснение строк: Первое значение в файле листинга - номер строки, и он может вовсе не совпадать с номером строки изначального файла. Второе вхождение - адрес, смещение машинного кода относительно начала текущего сегмента, затем непосредственно идет сам машинный код, а заключает строку исходный текст программы с комментариями.

3 Задание для самостоятельной работы

1. Напишем программу нахождения наименьшей из 3 целочисленных переменных a,b и . Значения переменных выберем из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создадим исполняемый файл и проверим его работу:

рис. 3.1:

```
asinova1:~/work/arch-pc/lab07$ nasm -f elf lab7-3.asm
asinova1:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
asinova1:~/work/arch-pc/lab07$/lab7-3
Наименьшее число: 17
asinova1:~/work/arch-pc/lab07$
```

Рисунок 3.1: Выполнение задания

рис. 3.2:

```
ASM NAME: 7.2
%include 'in_out.asm'

SECTION .data
a dd 44
b dd 74
c dd 17
min_msg db 'Наименьшее число: ',0

SECTION .text
GLOBAL _start
_start:
    mov eax, [a]
    mov ebx, [b]
    mov ecx, [c]

    cmp eax, ebx
    jl compare_with_c
    mov eax, ebx

    compare_with_c:
    cmp eax, ecx
    jl print_result
    mov eax, ecx

    print_result:
    mov esi, eax

    mov eax, min_msg
    call sprint

    mov eax, esi
    call iprintLF
    call quit
```

Рисунок 3.2: Выполнение задания

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

рис. 3.3:

```
%include 'in_out.asm'

SECTION .data
msg_x db 'Введите x: ',0
msg_a db 'Введите a: ',0
msg_result db 'Результат: ',0

SECTION .bss
x resd 1
a resd 1

SECTION .text
GLOBAL _start
_start:
    mov eax, msg_x
    call sprint
    call readint
    mov [x], eax

    mov eax, msg_a
    call sprint
    call readint
    mov [a], eax

    mov ebx, [x]
    mov ecx, [a]

    cmp ebx, 4
    jl case1

    case2:
    imul ebx, ecx
    jmp print_result

    case1:
        add ebx, 1
        imul ebx, ecx
        jmp print_result

print_result:
    mov eax, msg_result
    call sprint
    mov eax, ebx
    call iprint
```

Рисунок 3.3: Выполнение задания

рис. 3.4:

```
asinova1:~/work/arch-pc/lab07$ nasm -f elf lab7-4.asm
asinova1:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-4 lab7-4.o
asinova1:~/work/arch-pc/lab07$ ./lab7-4
Введите x: 1
Введите a: 1
Результат: 5
```

Рисунок 3.4: Выполнение задания

рис. 3.5:

```
asinova1:~/work/arch-pc/lab07$ nasm -f elf lab7-4.asm
asinova1:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-4 lab7-4.o
asinova1:~/work/arch-pc/lab07$ ./lab7-4
Введите x: 7
Введите a: 1
Результат: 7
```

Рисунок 3.5: Выполнение задания

рис. 3.6:

```
%include 'in_out.asm'
SECTION .data
msg_x db 'Введите x: ',0
msg_a db 'Введите a: ',0
msg_result db 'Результат: ',0
SECTION .bss
x resd 1
a resd 1
SECTION .text
GLOBAL _start
_start:
    mov eax, msg_x
    call sprint
    call readint
    mov [x], eax

    mov eax, msg_a
    call sprint
    call readint
    mov [a], eax

    mov ebx, [x]
    mov ecx, [a]

    cmp ebx, 4
    jl case1

case2:
    imul ebx, ecx
    jmp print_result

case1:
    mov eax, msg_result
    call sprint
print_result:
    mov eax, msg_result
    call sprint
```

Рисунок 3.6: Выполнение задания

4 Вывод

Мы изучили команды условного и безусловного перехода. Приобрели навыки написания программ с использованием переходов. Познакомились с назначением и структурой файла листинга.

Список литературы