

Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютеров

Симонова Виктория Игоревна

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 файла листинга	13
4.3	Задание для самостоятельной работы	15
5	Выводы	21
	Список литературы	22

Список иллюстраций

4.1	Создаю папку и файл	8
4.2	Изменяю файл	9
4.3	Запускаю исполняемый файл	9
4.4	Вывод исполняемого файла	9
4.5	Изменяю кодом из листинга	10
4.6	Вывод изменённого файла	10
4.7	Изменяю свой кодом	11
4.8	Вывод изменённого файла	11
4.9	Создаю файл	11
4.10	Вношу текст программы в файл	12
4.11	Вывод сравнения	13
4.12	Создаю файл листинга	13
4.13	Открываю файл	14
4.14	Строки файла листинга	14
4.15	Убираю операнд	15
4.16	Ошибка трансляции	15
4.17	Код программы	16
4.18	Вывод наименьшего числа	16
4.19	Программа для вычисления функции	18
4.20	Считаю выражение	19

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 файла листинга.
3. Задание 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.

-Команды безусловного перехода 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.

-Команды условного перехода Как отмечалось выше, для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов.

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

Создаю каталог lab07.asm для файлов лабораторной работы и файл lab7-1(рис. [4.1]).

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

Рис. 4.1: Создаю папку и файл

Ввожу в файл lab7-1.asm пример программы с использованием инструкции jmp (рис. [4.2]).


```

lab7-1.asm      [-M--] 41 L: [ 1+19 20/ 20] *(649 / 649b) <EOF>      [*][X]
#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: Изменяю файл

Создаю и запускаю исполняемый файл (рис. [4.3]).

```

[visimonova@fedora ~]$ mc

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

```

Рис. 4.3: Запускаю исполняемый файл

Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения. (рис. [4.4]).

```

Сообщение № 2
Сообщение № 3
[visimonova@fedora lab07]$

```

Рис. 4.4: Вывод исполняемого файла

Программа выводит Сообщение № 2 ,Сообщение № 3 .Изменяю файл так,чтобы программа выводила выводила а сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу.(рис. [4.5]).

```
%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. 4.5: Изменяю кодом из листинга

Создаю и запускаю исполняемый файл (рис. [4.6]).

```
[visimonova@fedora lab07]$ ./lab7_1
Сообщение № 2
Сообщение № 1
[visimonova@fedora lab07]$
```

Рис. 4.6: Вывод изменённого файла

Программа выводит Сообщение № 2 ,Сообщение № 1 .Изменяю файл так,чтобы программа выводила выводила а сначала ‘Сообщение № 3’, потом ‘Сообщение № 2’, ‘Сообщение № 1’ и завершала работу.(рис. [4.7]).

```

lab7-1.asm      [-M--] 11 L: [ 2+19 21/ 23] *(607 / 682)
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:
call quit ; вызов подпрограммы завершения

```

Рис. 4.7: Изменяю свой кодом

Создаю и запускаю исполняемый файл (рис. [4.8]).

```

[visimonova@fedora lab07]$ nasm -f elf lab7-1.asm
[visimonova@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[visimonova@fedora lab07]$ ./lab7-1
bash: ./lab7-1: Нет такого файла или каталога
[visimonova@fedora lab07]$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
[visimonova@fedora lab07]$

```

Рис. 4.8: Вывод изменённого файла

Создаю файл lab7-2.asm (рис. [4.9]).

```

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

```

Рис. 4.9: Создаю файл

Ввожу в него код из листинга для вывода наибольшей целочисленной переменной (рис. [4.10]).

```
lab7-2.asm [----] 17 L: [ 1+ 0 1/ 49] *(17 /1743b)
#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'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx,[max]
cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [max],ecx
```

Рис. 4.10: Вношу текст программы в файл

Создаю и запускаю исполняемый файл (рис. [4.11]).

```

[visimonova@fedora lab07]$ nasm -f elf lab7-2.asm
[visimonova@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[visimonova@fedora lab07]$ ./lab7-2
Введите B: 70
Наибольшее число: 70
[visimonova@fedora lab07]$

```

Рис. 4.11: Вывод сравнения

4.2 Изучение структуры файла листинга

Создаю файл листинга для программы из файла lab7-2.asm с помощью команды “nasm -f elf -l lab7-2.lst lab7-2.asm” (рис. [4.12]).

```

[visimonova@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm

```

Рис. 4.12: Создаю файл листинга

Откройте файл листинга lab7-2.lst с помощью текстового редактора (рис. [4.13]).

```

1                                     %include 'in_out.asm'
1                                     <1> ;----- slen -----
-----
2                                     <1> ; Функция вычисления длины сообщения
3                                     <1> slen:
4 00000000 53                         <1>     push     ebx
5 00000001 89C3                       <1>     mov      ebx, eax
6                                     <1>
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                                     <1>
13                                     <1> finished:
14 0000000B 29D8                      <1>     sub      eax, ebx
15 0000000D 5B                        <1>     pop      ebx
16 0000000E C3                        <1>     ret
17                                     <1>
18                                     <1>
19                                     <1> ;----- sprint -----
-----
20                                     <1> ; Функция печати сообщения
21                                     <1> ; входные данные: mov eax,<message>
22                                     <1> sprint:
23 0000000F 52                         <1>     push     edx
24 00000010 51                         <1>     push     ecx
25 00000011 53                         <1>     push     ebx
26 00000012 50                         <1>     push     eax
27 00000013 E8E8FFFFFF                 <1>     call     slen
28                                     <1>
29 00000018 89C2                      <1>     mov      edx, eax
30 0000001A 58                        <1>     pop      eax
31                                     <1>
32 0000001B 89C1                      <1>     mov      ecx, eax
33 0000001D BB01000000                 <1>     mov      ebx, 1
34 00000022 B804000000                 <1>     mov      eax, 4
35 00000027 CD80                      <1>     int      80h
36                                     <1>
37 00000029 5B                        <1>     pop      ebx
38 0000002A 59                        <1>     pop      ecx
39 0000002B 5A                        <1>     pop      edx

```

Рис. 4.13: Открываю файл

В данных трёх содержится следующее: (рис. [4.14]).

```

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

```

Рис. 4.14: Строки файла листинга

“2”, “3”, “4” -номера строчек.Строчки “2”, “3” не содержат адреса и машинного кода, т.к. в строке “2” записан комментарий, а в “3” название функции (в строке “4” 00000000-адрес строки, 53 машинный код).

Открываю файл с программой lab7-2.asm и в любой инструкции с двумя опе-

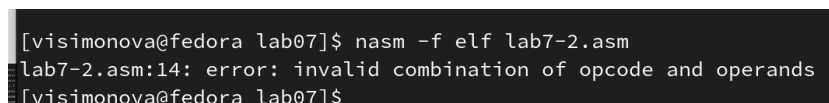
рандами удаляю один операнд. Выполняю трансляцию с получением файла листинга:(рис. [4.15]).



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

Рис. 4.15: Убираю операнд

В этом случае происходит ошибка трансляции (рис. [4.16]).



```
[visimonova@fedora lab07]$ nasm -f elf lab7-2.asm  
lab7-2.asm:14: error: invalid combination of opcode and operands  
[visimonova@fedora lab07]$
```

Рис. 4.16: Ошибка трансляции

Я не получаю ни одного файла, так как mov должка содержать два операнда.

4.3 Задание для самостоятельной работы

Задание 1 Создаю файл zadanie1.asm и ввожу в него код для сравнения трёх переменных и вывода наименьшего из них. (Значения переменных выбираю в соответствии с вариантом 13, полученным мной ранее)(рис. [4.17]).

```

%include 'in_out.asm'
section .data
msg2 db "Наименьшее число:",0h
A dd '84'
B dd '32'
C dd '77'
min resb 10
global _start
_start:
mov eax,C
call atoi ; Вызов подпрограммы перевода символа в число
mov [C],eax ; запись преобразованного числа в 'C'
; ----- Записываем 'A' в переменную 'min'
mov ecx,[A] ; 'ecx = A'
mov [min],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'B' (как символы)
cmp ecx,[B] ; Сравниваем 'A' и 'B'
jl check_C ; если 'A<B', то переход на метку 'check_C',
mov ecx,[B] ; иначе 'ecx = B'
mov [min],ecx ; 'min = B'
; ----- Преобразование 'max(A,C)' из символа в число
check_C:
mov eax,min
call atoi ; Вызов подпрограммы перевода символа в число
mov [min],eax ; запись преобразованного числа в min
; ----- Сравниваем 'min(A,B)' и 'C' (как числа)
mov ecx,[min]
cmp ecx,[C] ; Сравниваем 'min(A,B)' и 'C'
jl fin ; если 'min(A,B)<C', то переход на 'fin',
mov ecx,[C] ; иначе 'ecx = C'
mov [min],ecx
; ----- Вывод результата
fin:

```

Рис. 4.17: Код программы

Создаю и запускаю исполняемый файл, программа вывела наименьшее число из трёх(из 83,32,77 вывела 32) (рис. [4.18]).

```

[visimonova@fedora lab07]$ nasm -f elf zadanie1.asm
[visimonova@fedora lab07]$ ld -m elf_i386 -o zadanie1 zadanie1.o
[visimonova@fedora lab07]$ ./zadanie1
Наименьшее число:32

```

Рис. 4.18: Вывод наименьшего числа

Код к заданию 1:

```

%include 'in_out.asm'
section .data
msg2 db "Наименьшее число:",0h

```



```

A dd '84'
B dd '32'
C dd '77'
section .bss
min resb 10
section .text
global _start
_start:
mov eax,C
call atoi ; Вызов подпрограммы перевода символа в число
mov [C],eax ; запись преобразованного числа в 'C'
; ----- Записываем 'A' в переменную 'min'
mov ecx,[A] ; 'ecx = A'
mov [min],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'B' (как символы)
cmp ecx,[B] ; Сравниваем 'A' и 'B'
jnl check_C ; если 'A<B', то переход на метку 'check_C',
mov ecx,[B] ; иначе 'ecx = B'
mov [min],ecx ; 'min = B'
; ----- Преобразование 'max(A,C)' из символа в число
check_C:
mov eax,min
call atoi ; Вызов подпрограммы перевода символа в число
mov [min],eax ; запись преобразованного числа в min
; ----- Сравниваем 'min(A,B)' и 'C' (как числа)
mov ecx,[min]
cmp ecx,[C] ; Сравниваем 'min(A,B)' и 'C'
jnl fin ; если 'min(A,B)<C', то переход на 'fin',
mov ecx,[C] ; иначе 'ecx = C'

```

```

mov [min],ecx
; ----- Вывод результата
fin:
mov eax, msg2
call sprint ; Вывод сообщения 'Наименьшее число: '
mov eax,[min]
call iprintfLF ; Вывод 'min(A,B,C)'
call quit ; Выход

```

Задание 2

Создаю файл zadanie2.asm и ввожу в него код для программы, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции f(x)(вариант 13) и выводит результат вычислений.(рис. [4.19]).

```

; /home/vladimir/works/pc/zadanie2.asm
%include 'in_out.asm'
section .data
msg1 db "Введите x:",0h
msg2 db "Введите a:",0h
msg3 db "Ответ:",0h
section .bss
x resb 10
a resb 10
section .text
global _start
_start:
mov eax,msg1
call sprint
mov ecx,x
mov edx,10
call sread
mov eax,x
call atoi ; Вызов подпрограммы перевода символа в число
mov [x],eax ; запись преобразованного числа в 'x'

mov eax,msg2
call sprint

```

Рис. 4.19: Программа для вычисления функции

Создаю и запускаю исполняемый файл, в обоих вариантах введенного числа а программа работает корректно и вычисляет значения для двух соответствующих случаев функции. (рис. [4.20]).

```

[visimonova@fedora lab07]$ nasm -f elf zadanie2.asm
[visimonova@fedora lab07]$ ld -m elf_i386 -o zadanie2 zadanie2.o
[visimonova@fedora lab07]$ ./zadanie2
Введите x:3
Введите a:9
Ответ:2
[visimonova@fedora lab07]$ ./zadanie2
Введите x:6
Введите a:4
Ответ:24

```

Рис. 4.20: Считаю выражение

Код к заданию 2:

```

#include 'in_out.asm'

section .data
msg1 db "Введите x:",0h
msg2 db "Введите a:",0h
msg3 db "Ответ:",0h

section .bss
x resb 10
a resb 10

section .text
global _start
_start:
mov eax,msg1
call sprint
mov ecx,x
mov edx,10
call sread
mov eax,x
call atoi ; Вызов подпрограммы перевода символа в число
mov [x],eax ; запись преобразованного числа в 'x'

```

```

mov eax,msg2
call sprint
mov ecx,a
mov edx,10
call sread
; ----- Преобразование 'a' из символа в число
mov eax,a
call atoi ; Вызов подпрограммы перевода символа в число
mov [a],eax ; запись преобразованного числа в 'a'

mov eax,[a]
cmp eax,7
jl fin
jmp fin1
fin:
mov eax, msg3
call sprint
mov eax,[a]
mov ebx,[x]
mul ebx
call iprintLF
call quit ; Выход
fin1:
mov eax,msg3
call sprint
mov eax,[a]
add eax,-7
call iprintLF
call quit ; Выход

```

5 Выводы

Изучила команды условного и безусловного перехода, научилась писать программы с использованием переходов . Познакомилась со структурой и назначением файла листинга.

Список литературы

Лабораторная работа №7