

Лабораторная работа номер 7

Архитектура компьютера

Титков Ярослав Максимович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Реализация переходов в NASM:	9
4.2	Изучение структуры файла листинга:	11
4.3	Ответы на вопросы:	11
4.4	Задания для самостоятельной работы:	13
5	Выводы	15

Список иллюстраций

4.1	Создал каталог lab07, затем перешёл в него и создал файл lab7-1.asm	9
4.2	Вставил нужную функцию и проверил из Листинга 7.1 и запустил	9
4.3	Изменил программу lab7-1.asm из Листинга 7.2 и запустил её . . .	10
4.4	Создал файл lab7-2.asm и внёс туда данные из Листинга 7.3	10
4.5	Запустил программу lab7-2	10
4.6	Получил файл Листинга и открыл в mcedit	11
4.7	Выполнил трансляцию с получением файла листинга	11
4.8	Создал файл lab7-3.asm и написал программу с данными из 15 варианта	13
4.9	Запустил программу	13
4.10	Аналогично заданию 1 создал программу lab7-4.asm, которая выводит результат вычислений $f(x)$ с переменными x и a с данными из 15 варианта	14
4.11	Запустил, введя данные 2 и 3, 4 и 2	14

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. Задание для самостоятельной работы

3 Теоретическое введение

Для реализации ветвлений в ассемблере используются так называемые команды передачи управления или команды перехода. Существует два типа переходов: условный и безусловный. Безусловный переход выполняется инструкцией `jmp` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление: `jmp <адрес_перехода>`. Адрес перехода может быть либо меткой, либо адресом области памяти, в которую предварительно помещен указатель перехода. В качестве операнда можно также использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре.

Условный переход, в отличие от безусловного, требует проверки определенного условия. В ассемблере команды условного перехода анализируют флаги из регистра флагов для определения необходимости перехода. Флаг – это бит, принимающий значение 1 («флаг установлен»), если выполнено некоторое условие, и значение 0 («флаг сброшен») в противном случае. Флаги работают независимо друг от друга и помещены в единый регистр – регистр флагов, отражающий текущее состояние процессора.

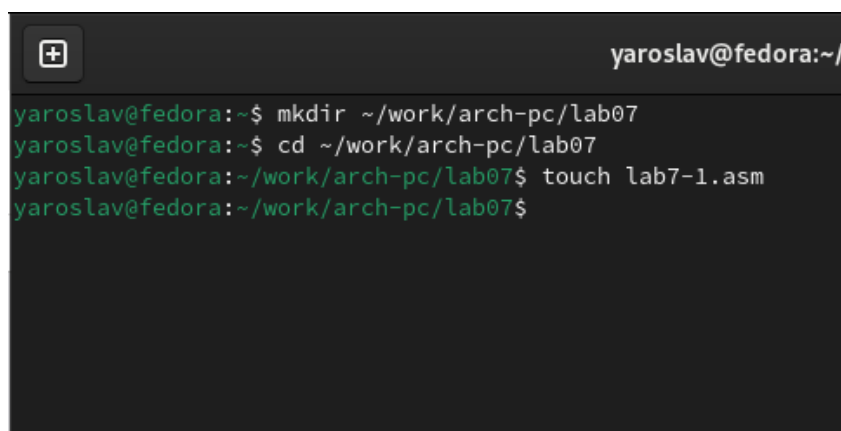
Инструкция `cmp` является одной из команд, которая позволяет сравнить операнды и выставить флаги в зависимости от результата сравнения. Формат команды `cmp` аналогичен команде вычитания: `cmp <операнд_1>, <операнд_2>`. Команда `cmp` выполняет вычитание `<операнд_2> - <операнд_1>`, но результат вычитания никуда не записывается, и единственным результатом команды сравнения является формирование флагов.

Команда условного перехода имеет вид `j<мнемоника перехода> label`, где мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов. В таблице представлены команды условного перехода, которые обычно ставятся после команды сравнения `cmp`. В их мнемониках указывается тот результат сравнения, при котором надо делать переход. Мнемоники, идентичные по своему действию, написаны в таблице через дробь (например, `ja` и `jnb`). Программист выбирает, какую из них применить, чтобы получить более простой для понимания текст программы.

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. Ниже приведён фрагмент файла листинга.

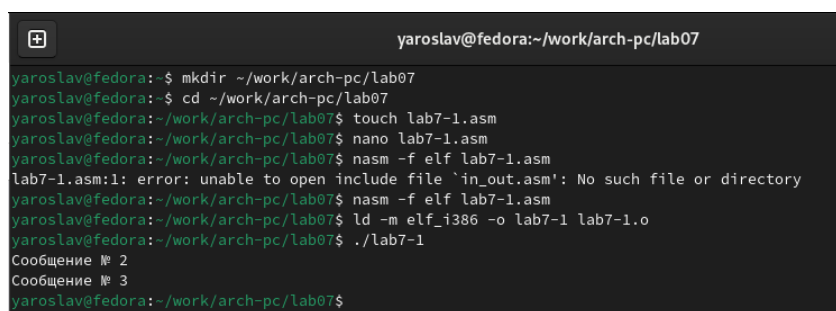
4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM:



```
yaroslav@fedora:~$ mkdir ~/work/arch-pc/lab07
yaroslav@fedora:~$ cd ~/work/arch-pc/lab07
yaroslav@fedora:~/work/arch-pc/lab07$ touch lab7-1.asm
yaroslav@fedora:~/work/arch-pc/lab07$
```

Рис. 4.1: Создал каталог lab07, затем перешёл в него и создал файл lab7-1.asm



```
yaroslav@fedora:~/work/arch-pc/lab07$ touch lab7-1.asm
yaroslav@fedora:~/work/arch-pc/lab07$ nano lab7-1.asm
yaroslav@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
lab7-1.asm:1: error: unable to open include file 'in_out.asm': No such file or directory
yaroslav@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
yaroslav@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
yaroslav@fedora:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
yaroslav@fedora:~/work/arch-pc/lab07$
```

Рис. 4.2: Вставил нужную функцию и проверил из Листинга 7.1 и запустил

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

Рис. 4.3: Изменил программу lab7-1.asm из Листинга 7.2 и запустил её

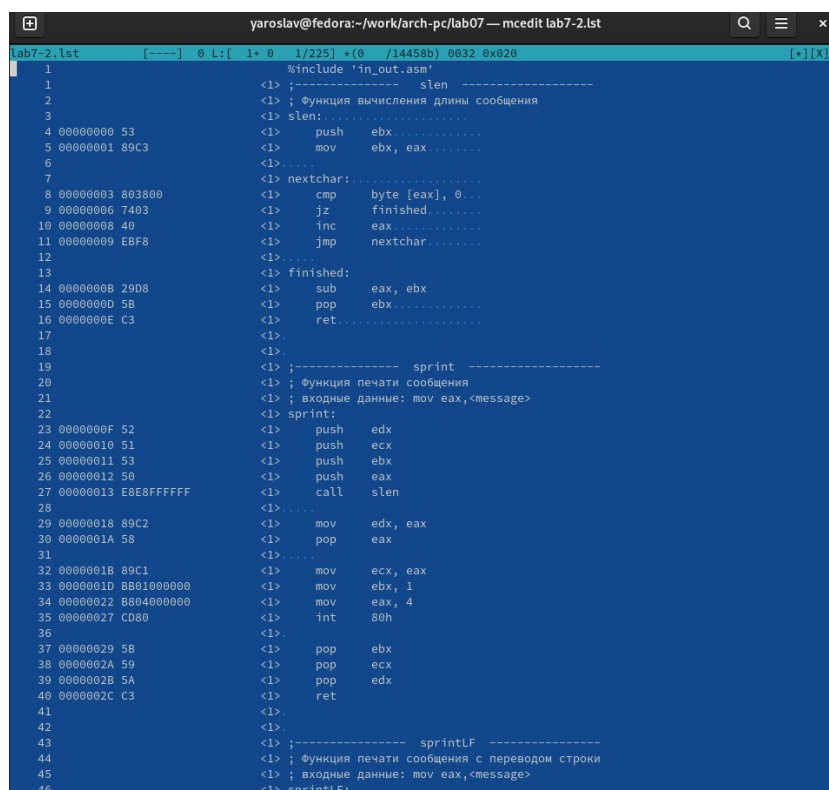
```
GNU nano 7.2 lab7-2.asm
#include "in_out.asm"
section .data
msg1 db 'Введите B: ',0h
msg2 db "Наибольшее число: ",0h
A dd '20'
C dd '50'
section .bss
max resb 10
B resb 10
section .text
global _start
_start:
; ----- Вывод сообщения 'Введите B: '
mov eax,msg1
call sprint
; ----- Ввод 'B'
mov ecx,B
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
; ----- Вывод результата
fin:
mov eax, msg2
[ Прочитано 49 строк ]
```

Рис. 4.4: Создал файл lab7-2.asm и внёс туда данные из Листинга 7.3

```
yaroslav@fedora:~/work/arch-pc/lab07
yaroslav@fedora:~/work/arch-pc/lab07$ touch lab7-2.asm
yaroslav@fedora:~/work/arch-pc/lab07$ nano lab7-2.asm
yaroslav@fedora:~/work/arch-pc/lab07$ nano lab7-2.asm
yaroslav@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
yaroslav@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
yaroslav@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 1
Наибольшее число: 50
yaroslav@fedora:~/work/arch-pc/lab07$
```

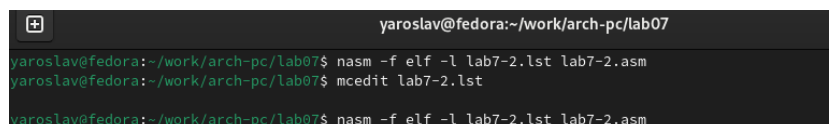
Рис. 4.5: Запустил программу lab7-2

4.2 Изучение структуры файла листинга:



```
lab7-2.lst [-----] 0 L:[ 1+ 0 1/225] +@ /14458b) 0032 0x020 [+][X]
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>----- sprint -----
19     <1> ; Функция печати сообщения
20     <1> ; входные данные: mov eax,<message>
21     <1> sprint:-----
22     <1>-----
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 B801000000 <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
40     0000002C C3      <1> ret   -----
41     <1>-----
42     <1>----- sprintLF -----
43     <1> ; Функция печати сообщения с переводом строки
44     <1> ; входные данные: mov eax,<message>
45     <1> sprintLF:-----
46     <1>-----
```

Рис. 4.6: Получил файл Листинга и открыл в mcedit



```
yaroslav@fedora:~/work/arch-pc/lab07
yaroslav@fedora:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
yaroslav@fedora:~/work/arch-pc/lab07$ mcedit lab7-2.lst
yaroslav@fedora:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
```

Рис. 4.7: Выполнил трансляцию с получением файла листинга

4.3 Ответы на вопросы:

1 -f elf: Указывает формат объектного файла (ELF). -l lab7-2.lst: Указывает имя файла листинга, который будет создан. lab7-2.asm: Исходный файл ассемблера, который будет ассемблирован.

2 Формат и содержимое файла листинга:

файл листинга (lab7-2.lst) содержит следующие основные элементы:

Номер строки: Указывает номер строки в исходном файле.

Адрес: Указывает адрес команды в памяти.

Машинный код: Отображает машинный код, соответствующий команде.

Исходный код: Отображает исходную строку ассемблера.

3

1. 10 00000000 B800000000 mov eax, 0:

10 Номер строки в исходном файле. 00000000: Адрес команды в памяти.

B800000000: Машинный код команды mov eax, 0. mov eax, 0: Исходная строка ассемблера, которая загружает значение 0 в регистр eax.

2. 15 00000005 89C3 mov ebx, eax: 0: Номер строки в исходном файле. 00000000:

Адрес команды в памяти. B800000000: Машинный код команды mov eax, 0. mov eax, 0: Исходная строка ассемблера, которая загружает значение 0 в регистр eax.

3. 20 00000007 CD80 int 0x80: 20: Номер строки в исходном файле. 00000007:

Адрес команды в памяти. CD80: Машинный код команды int 0x80. int 0x80: Исходная строка ассемблера, которая вызывает прерывание 0x80 для выполнения системного вызова.

4

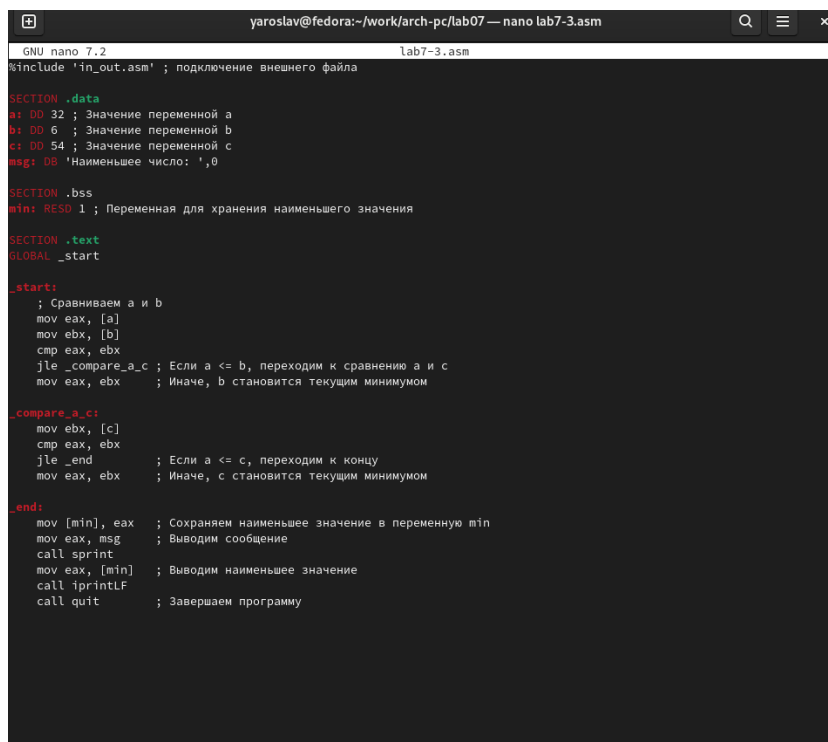
1. Что создаётся:

Файл листинга (lab7-2.lst): Этот файл будет создан, но он будет содержать информацию об ошибке, так как трансляция не может быть успешно завершена из-за неполной инструкции. Объектный файл (lab7-2.o): Этот файл не будет создан, так как трансляция завершится с ошибкой.

2. Что добавляется: В файле листинга (lab7-2.lst) будет добавлена информация об ошибке, указывающая на неполную инструкцию

4.4 Задания для самостоятельной работы:

1. Напишите программу нахождения наименьшей из 3 целочисленных переменных a, b и c. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу.



```
GNU nano 7.2 lab7-3.asm
#include "in_out.asm" ; подключение внешнего файла

SECTION .data
a: DD 32 ; Значение переменной a
b: DD 6 ; Значение переменной b
c: DD 54 ; Значение переменной c
msg: DB "Наименьшее число: ",0

SECTION .bss
min: RESD 1 ; Переменная для хранения наименьшего значения

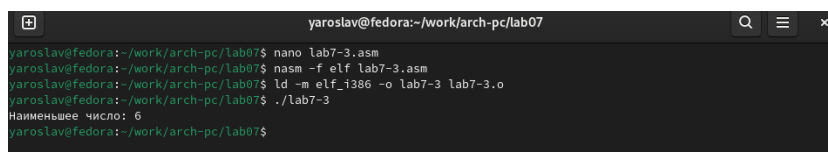
SECTION .text
GLOBAL _start

_start:
; Сравниваем a и b
mov eax, [a]
mov ebx, [b]
cmp eax, ebx
jle _compare_a_c ; Если a <= b, переходим к сравнению a и c
mov eax, ebx ; Иначе, b становится текущим минимумом

_compare_a_c:
mov ebx, [c]
cmp eax, ebx
jle _end ; Если a <= c, переходим к концу
mov eax, ebx ; Иначе, c становится текущим минимумом

_end:
mov [min], eax ; Сохраняем наименьшее значение в переменную min
mov eax, msg ; Выводим сообщение
call sprint
mov eax, [min] ; Выводим наименьшее значение
call iprintLF
call quit ; Завершаем программу
```

Рис. 4.8: Создал файл lab7-3.asm и написал программу с данными из 15 варианта



```
yaroslav@fedora:~/work/arch-pc/lab07$ nano lab7-3.asm
yaroslav@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-3.asm
yaroslav@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
yaroslav@fedora:~/work/arch-pc/lab07$ ./lab7-3
Наименьшее число: 6
yaroslav@fedora:~/work/arch-pc/lab07$
```

Рис. 4.9: Запустил программу

2. Напишите программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции f(x) и выводит результат вычислений. Вид функции f(x) выбрать из таблицы 7.6 вариантов заданий в

соответствии с вариантом, полученным при выполнении лабораторной работы номер 7. Создайте исполняемый файл и проверьте его работу для значений x и a из 7.6.

```
GNU nano 7.2 lab7-4.asm
%include 'in_out.asm'

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

section .bss
    x resb 10
    a resb 10
    result resb 10

section .text
    global _start

_start:
    ; Вывод сообщения для ввода x
    mov eax, msg_x
    call sprint

    ; Ввод значения x
    mov ecx, x
    mov edx, 10
    call sread

    ; Преобразование x из символа в число
    mov eax, x
    call atoi
    mov [x], eax

    ; Вывод сообщения для ввода a
    mov eax, msg_a
    call sprint

    ; Ввод значения a
    mov ecx, a
    mov edx, 10
    call sread

    ; Преобразование a из символа в число
    mov eax, a
    call atoi
    mov [a], eax

    ; Вычисление f(x)
```

Рис. 4.10: Аналогично заданию 1 создал программу lab7-4.asm, которая выводит результат вычислений $f(x)$ с переменными x и a с данными из 15 варианта

```
yaroslav@fedora:~/work/arch-pc/lab07 — nano lab7-4.asm × yaroslav@fedora:~/work/arch-pc/lab07
yaroslav@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-4.asm
yaroslav@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-4 lab7-4.o
yaroslav@fedora:~/work/arch-pc/lab07$ ./lab7-4
Введите x: 2
Введите a: 3
Результат: 13
yaroslav@fedora:~/work/arch-pc/lab07$ ./lab7-4
Введите x: 4
Введите a: 2
Результат: 14
yaroslav@fedora:~/work/arch-pc/lab07$ ./lab7-4
```

Рис. 4.11: Запустил, введя данные 2 и 3, 4 и 2

5 Выводы

В ходе выполнения лабораторной работы были изучены команды условного и безусловного переходов в ассемблере, что позволило приобрести навыки написания программ с использованием переходов. Команды переходов являются ключевыми для управления потоком выполнения программы, что позволяет реализовывать сложные алгоритмы и логические конструкции.