

Отчёт по лабораторной работе №7

Дисциплина: Архитектура компьютера

Долгаев Евгений Сергеевич НММбд-01-24

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
3.0.1	Команда безусловного перехода	7
3.0.2	Команда условного перехода	8
3.0.3	Регистр флагов	8
3.0.4	Описание инструкции cmp	11
3.0.5	Описание команд условного перехода.	12
3.1	Файл листинга и его структура	14
4	Выполнение лабораторной работы	16
4.1	Задания для самостоятельной работы	20
5	Выводы	23
	Список литературы	24

Список иллюстраций

3.1	Регистр флагов	9
3.2	Структура листинга	15
4.1	Подготовка рабочего пространства	16
4.2	Текст программы	16
4.3	Создание и работа исполняемого файла	17
4.4	Текст программы	17
4.5	Создание и работа исполняемого файла	18
4.6	Текст программы	18
4.7	Создание и работа исполняемого файла	18
4.8	Создание файла	19
4.9	Текст программы	19
4.10	Создание и работа исполняемого файла	19
4.11	Создание файла	19
4.12	Файл листинга	20
4.13	Создание файла с ошибкой	20
4.14	Файл листинга	20
4.15	Текст программы	21
4.16	Создание и работа исполняемого файла	21
4.17	Текст программы	21
4.18	Создание и работа исполняемого файла	22

Список таблиц

3.1	Типы операндов инструкции jmp	8
3.2	Регистр флагов	9
3.3	Инструкции условной передачи управления по результатам арифметического сравнения cmp a,b	12
3.4	Инструкции условной передачи управления	14

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) Выполнение лабораторной работы
 - 1) Реализация переходов в NASM
 - 2) Изучение структуры файлы листинга
- 2) Задания 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.

3.0.1 Команда безусловного перехода

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. Кроме того, в качестве операнда можно использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре (табл. 3.1).

Таблица 3.1: Типы операндов инструкции jmp

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

В следующем примере рассмотрим использование инструкции jmp:

```
label:
... ;
... ; команды
... ;
jmp label
```

3.0.2 Команда 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.

3.0.3 Регистр флагов

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 лишь для удобства они помещены в единый регистр – регистр флагов, отражающий текущее состояние процессора. В следующей таблице указано положение битовых флагов в регистре флагов (рис. 3.1).

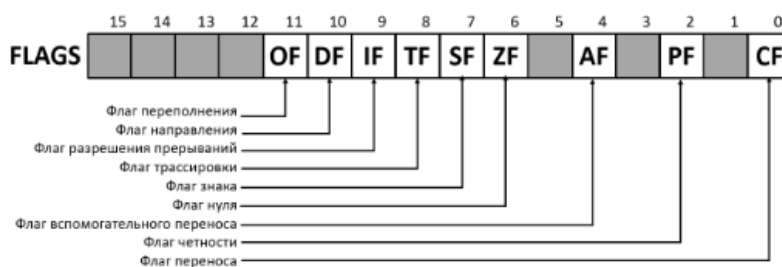


Рис. 3.1: Регистр флагов

Флаги состояния (биты 0, 2, 4, 6, 7 и 11) отражают результат выполнения арифметических инструкций, таких как ADD, SUB, MUL, DIV.

Таблица 3.2: Регистр флагов

Бит	Обозначение	Название	Описание
0	CF	Carry Flag - Флаг переноса	Устанавливается в 1, если при выполнении предыдущей операции произошёл перенос из старшего бита или если требуется заём (при вычитании). Иначе установлен в 0.

Бит	Обозначение	Название	Описание
2	PF	Parity Flag - Флаг чётности	Устанавливается в 1, если младший байт результата предыдущей операции содержит чётное количество битов, равных 1.
4	AF	Auxiliary Carry Flag - Вспомогательный флаг переноса	Устанавливается в 1, если в результате предыдущей операции произошёл перенос (или заём) из третьего бита в четвёртый.
6	ZF	Zero Flag - Флаг нуля	Устанавливается в 1, если результат предыдущей команды равен 0.

Бит	Обозначение	Название	Описание
7	SF	Sign Flag - Флаг знака	Равен значению старшего значащего бита результата, который является знаковым битом в знаковой арифметике.
11	SF	Overflow Flag - Флаг переполнения	Устанавливается в 1, если целочисленный результат слишком длинный для размещения в целевом операнде (регистре или ячейке памяти).

3.0.4 Описание инструкции `cmp`

Инструкция `cmp` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения.

Инструкция `cmp` является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания:

`cmp` <операнд_1>, <операнд_2>

Команда `cmp`, так же как и команда вычитания, выполняет вычитание

<операнд_2> - <операнд_1>, но результат вычитания никуда не записывается и единственным результатом команды сравнения является формирование флагов.

Примеры.

сmp ax, '4' ; сравнение регистра *ax* с символом 4

сmp ax, 4 ; сравнение регистра *ax* со значением 4

сmp al, cl ; сравнение регистров *al* и *cl*

сmp [buf], ax ; сравнение переменной *buf* с регистром *ax*

3.0.5 Описание команд условного перехода.

Команда условного перехода имеет вид

j<мнемоника перехода> label

Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов.

В табл. 3.3 представлены команды условного перехода, которые обычно ставятся после команды сравнения *сmp*. В их мнемокодах 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.

Таблица 3.3: Инструкции условной передачи управления по результатам арифметического сравнения *сmp a,b*

		Критерий условного перехода а	
Типы операндов	Мнемокод	(<,>,<=,>=,=) b	Комментарий
Любые	JE	a = b	ZF = 1

Типы операндов	Мнемокод	Критерий условного перехода а	
		(<,>,<=,>=,=) b	Комментарий
Любые	JNE	not(a = b)	ZF = 0
Со знаком	JL/JNGE	a < b	not(SF = OF)
Со знаком	JLE/JNG	a <= b	not(SF = OF) или ZF = 1
Со знаком	JG/JNLE	a > b	SF = OF и ZF = 0
Со знаком	JGE/JNL	a >= b	SF = OF
Без знака	JB/JNAE	a < b	CF = 1
Без знака	JBE/JNA	a <= b	CF = 1 или ZF = 1
Без знака	JA/JNBE	a > b	CF = 0 и ZF = 0
Без знака	JAЕ/JNB	a >= b	CF = 0

Примечание: термины «выше» («a» от англ. «above») и «ниже» («b» от англ. «below») применимы для сравнения беззнаковых величин (адресов), а термины «больше» («g» от англ. «greater») и «меньше» («l» от англ. «lower») используются при учёте знака числа. Таким образом, мнемонику инструкции JA/JNBE можно расшифровать как *jump if above* (переход если выше) / *jump if not below equal* (переход если не меньше или равно)».

Помимо перечисленных команд условного перехода существуют те, которые можно использовать после любых команд, меняющих значения флагов (табл. 3.4).

Таблица 3.4: Инструкции условной передачи управления

Мnemonic	Значение флага	Мnemonic	Значение флага
	для осуществления перехода		для осуществления переход
JZ	ZF = 1	JNZ	ZF = 0
JS	SF = 1	JNS	SF = 0
JC	CF = 1	JNC	CF = 0
JO	OF = 1	JNO	OF = 0
JP	PF = 1	JNP	PF = 0

В качестве примера рассмотрим фрагмент программы, которая выполняет умножение переменных `a` и `b` и если произведение превосходит размер байта, передает управление на метку `Error`.

```

mov al, a
mov bl, b
mul bl
jc Error

```

3.1 Файл листинга и его структура

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.

```

10 00000000 B804000000 mov eax,4
11 00000005 BB01000000 mov ebx,1

```

```

12 0000000A B9[00000000] mov ecx,hello
13 0000000F BA0D000000 mov edx,helloLen
14
15 00000014 CD80 int 80h

```

Строки в первой части листинга имеют следующую структуру (рис. 3.2).



Рис. 3.2: Структура листинга

Все ошибки и предупреждения, обнаруженные при ассемблировании, транслятор выводит на экран, и файл листинга не создаётся.

Итак, структура листинга:

- *номер строки* — это номер строки файла листинга (нужно помнить, что номер строки в файле листинга может не соответствовать номеру строки в файле с исходным текстом программы);
- *адрес* — это смещение машинного кода от начала текущего сегмента;
- *машинный код* представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности. (например, инструкция `int 80h` начинается по смещению `00000020` в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция `int 80h` ассемблируется в `CD80` (в шестнадцатеричном представлении); `CD80` — это инструкция на машинном языке, вызывающая прерывание ядра);
- *исходный текст программы* — это просто строка исходной программы вместе с комментариями (некоторые строки на языке ассемблера, например, строки, содержащие только комментарии, не генерируют никакого машинного кода, и поля «смещение» и «исходный текст программы» в таких строках отсутствуют, однако номер строки им присваивается).

4 Выполнение лабораторной работы

Создим каталог для программ лабораторной работы № 7(рис. 4.1)

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

Рис. 4.1: Подготовка рабочего пространства

Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`. Введём в файл `lab7-1.asm` текст программы(рис. 4.2)

```
/home/es-b7-1.asm [---] 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: Текст программы

Создим исполняемый файл и запустим его. Результат работы данной программы будет следующим(рис. 4.3):


```

esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 la
b7-1.o
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$

```

Рис. 4.3: Создание и работа исполняемого файла

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 № 2', потом 'Сообщение № 1' и завершала работу. Для этого в текст программы после вывода сообщения № 2 добавим инструкцию `jmp` с меткой `_label1` (т.е. переход к инструкциям вывода сообщения № 1) и после вывода сообщения № 1 добавим инструкцию `jmp` с меткой `_end` (т.е. переход к инструкции `call quit`)(рис. 4.4).

```

/home/es-b7-1.asm [----] 41 L:[ 1+21 22/ 22] *(670 / 670b) <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'
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. 4.4: Текст программы

Создим исполняемый файл и проверим его работу(рис. 4.5).

```

esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 la
b7-1.o
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1

```

Рис. 4.5: Создание и работа исполняемого файла

Изменим текст программы добавив или изменив инструкции `jmp`, чтобы вывод програм- мы был следующим(рис. 4.6):

Сообщение № 3

Сообщение № 2

Сообщение № 1

```

/home/es-b7-1.asm [----] 11 L:[ 1+20 21/ 23] *(607 / 682b) 0010 0x00A [*][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 _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:

```

Рис. 4.6: Текст программы

Создим исполняемый файл и проверим его работу(рис. 4.7).

```

esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 la
b7-1.o
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1

```

Рис. 4.7: Создание и работа исполняемого файла

Создим файл lab7-2.asm в каталоге ~/work/arch-pc/lab07 и введём в него текст программы(рис. 4.8, 4.9).

```
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ touch lab7-2.asm
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ls
in_out.asm lab7-1 lab7-1.asm lab7-1.o lab7-2.asm
```

Рис. 4.8: Создание файла

```
/home/esdolgaev/work/arch-pc/lab07/lab7-2.asm [----] 11 L: 1+ 9 10/ 49] *(181 /1743b) 0120 0x078 [*][X]
%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'
1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перем-тить 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 4.9: Текст программы

Создим исполняемый файл и проверим его работу(рис. 4.10).

```
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 la
b7-2.o
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 10
Наибольшее число: 50
```

Рис. 4.10: Создание и работа исполняемого файла

Создим файл листинга для программы из файла lab7-2.asm и откроем его с помощью любого текстового редактора mcedit(рис. 4.11, 4.12).

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

Рис. 4.11: Создание файла

```

/home/esdolgaev/work/arch-pc/lab07/lab7-2.lst [----] 61 L: [ 1+ 0 1/225] *(61 /14458b) 0010 0x00A [*][X
1                                     %include 'in_out.asm'
2                                     <1> ;----- slen -----
3                                     <1> ; Функция вычисления длины сообщения
4 00000000 53                         <1> slen:
5 00000001 89C3                       <1>     push    ebx
6                                     <1>     mov     ebx, eax
7                                     <1> nextchar:
8 00000003 003000                     <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 2908                      <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

```

Рис. 4.12: Файл листинга

Откроем файл с программой lab7-2.asm и в любой инструкции с двумя операндами удалим один операнд. Выполним трансляцию с получением файла листинга(рис. 4.13, 4.14).

```

esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:34: error: invalid combination of opcode and operands

```

Рис. 4.13: Создание файла с ошибкой

```

32                                     ; ----- Преобразование 'max(A,C)' из символа в число
33                                     check_B:
34                                     mov     eax,
35                                     *****
36 00000130 E867FFFFFF                call    atoi ; Вызов подпрограммы перевода символа в число
37 00000135 A3[00000000]              mov     [max],eax ; запись преобразованного числа в 'max'
                                     ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)

```

Рис. 4.14: Файл листинга

Файл создавался тот же, но в нём появилась пометка об ошибке.

4.1 Задания для самостоятельной работы

Создадим копию файла lab7-2.asm и введём в него текст программы которая находит наименьшее из трёх чисел. Создадим исполняемый файл и проверим его работу(рис. 4.13, 4.14).

```

/home/esdolgaev/work/arch-pc/lab07/variant8.asm [----] 50 L: [ 1+21 22/ 49] *(526 /1743b) 1095 0x447 [*][X]
%include 'ln_out.asm'
section .data
msg1 db 'Введите B: ',0h
msg2 db 'Наименьшее число: ',0h
A dd '52'
C dd '33'
section .bss
min 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,[C] ; 'ecx = A'
mov [min],ecx ; 'min = 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'
1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перем-титы 7Поиск 8Удалить 9МенюМС 10Выход

```

Рис. 4.15: Текст программы

```

esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf variant8.asm
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o variant8 variant8.o
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ./variant8
Введите B: 40
Наименьшее число: 33
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$

```

Рис. 4.16: Создание и работа исполняемого файла

Далее напишем программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции $f(x)$ и выводит результат вычислений(рис. 4.17, 4.18).

```

/home/esdolgaev/work/arch-pc/lab07/variant8-1.asm [----] 0 L: [ 24+ 1 25/ 53] *(342 / 737b) 0109 0x060 [*][X]
; Ввод a
mov eax, msg2
call sprint
mov ecx, a
mov edx, 10
call sread
mov eax, a
call atoi
mov [a], eax
; Вычисление f(x)
mov eax, [a]
cmp eax, 3
jl calc_3x
jmp calc_a1
calc_3x:
mov eax, [a]
imul eax, 3
mov [res], eax
jmp print_result
calc_a1:
mov eax, [x]
add eax, 1
mov [res], eax
print_result:
; Вывод результата
mov eax, msg3
call sprint
mov eax, [res]
call tprintf
call quit

```

Рис. 4.17: Текст программы

```
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf variant8-1.asm
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o variant8-1 variant8-1.o

esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ./variant8-1
Введите x: 1
Введите a: 4
Результат f(x): 2
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ./variant8-1
Введите x: 1
Введите a: 2
Результат f(x): 6
```

Рис. 4.18: Создание и работа исполняемого файла

5 Выводы

В ходе выполнения лабораторной работы, я изучил команды условного и безусловного переходов, приобрёл навыки написания программ с использованием переходов и познакомился с назначением и структурой файла листинга.

Список литературы