

Отчет по лабораторной работе 8

**Команды безусловного и условного переходов в Nasm.
Программирование ветвлений.**

Гисматуллин Артём Вадимович НПИбд-01-22

Содержание

| | | |
|----------|--|-----------|
| 1 | Цель работы | 5 |
| 2 | Задание | 6 |
| 3 | Теоретическое введение | 7 |
| 3.1 | Команды безусловного перехода | 7 |
| 3.2 | Команды условного перехода | 7 |
| 3.2.1 | Флаги | 8 |
| 3.2.2 | Описание инструкции cmp | 8 |
| 3.2.3 | Описание команд условного перехода. | 9 |
| 3.3 | Файл листинга и его структура | 10 |
| 4 | Выполнение лабораторной работы | 12 |
| 5 | Выполнение задания для самостоятельной работы | 18 |
| 6 | Выводы | 23 |
| | Список литературы | 24 |

Список иллюстраций

| | | |
|-----|--|----|
| 3.1 | Регистр флагов. | 8 |
| 3.2 | Арифметическое сравнение стр a и b | 10 |
| 4.1 | Командная строка. каталог ~/work/arch-pc/lab08 | 12 |
| 4.2 | Командная строка. Выполнение программы lab8-1.asm | 13 |
| 4.3 | Командная строка. Выполнение измененной программы lab8-1.asm | 14 |
| 4.4 | Командная строка. Выполнение измененной программы lab8-1.asm 2 | 15 |
| 4.5 | Командная строка. Выполнение программы lab8-2.asm 2 | 16 |
| 4.6 | Командная строка. Просмотр содержимого файла листинга | 16 |
| 4.7 | Командная строка. Просмотр содержимого измененного файла ли- стинга | 17 |
| 5.1 | Командная строка. Просмотр содержимого файла lab8-3.asm | 19 |
| 5.2 | Командная строка. Выполнение программы lab8-3.asm | 20 |
| 5.3 | Командная строка. Просмотр содержимого файла lab8-4.asm | 21 |
| 5.4 | Командная строка. Выполнение программы lab8-4.asm | 22 |

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. Реализовать безусловный переход с помощью листинга 8.1. Выполнить соответствующее задание с выводом сообщений в определенном порядке.
2. Реализовать программу листинга 8.3, выводящую на экран наибольшее из трех целочисленных значений.
3. Изучить структуру файлов листинга. Ответить на вопросы.
4. Реализовать программу по нахождению наименьшего из трех целочисленных значений.
5. Написать программу, которая будет вычислять значение функции с определенным условием и данными значениями переменных (по вариантам).

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.

3.1 Команды безусловного перехода

Безусловный переход выполняется инструкцией `jmp`, 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.

3.2 Команды условного перехода

В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов.

3.2.1 Флаги

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)

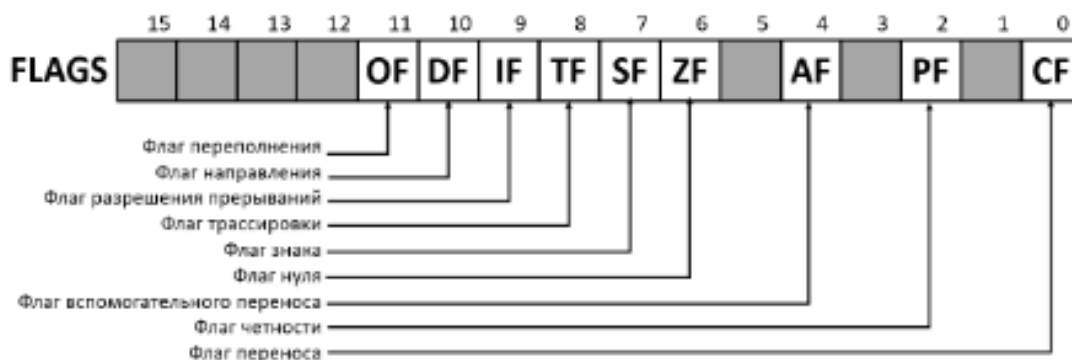


Рис. 3.1: Регистр флагов.

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

3.2.2 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 ,`

Команда `cmp`, так же как и команда вычитания, выполняет вычитание -, но результат вычитания никуда не записывается и единственным результатом команды сравнения является формирование флагов.

3.2.3 Описание команд условного перехода.

Команда условного перехода имеет вид

`j label`

Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов. В таблице (рис. 3.2) представлены некоторые команды условного перехода, которые обычно ставятся после команды сравнения `cmp`. 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.

| Типы операндов | Мнемокод | Критерий условного перехода $a \vee b$ | Значения флагов | Комментарий |
|----------------|----------|--|-------------------------|-------------------------------|
| Любые | JE | $a = b$ | ZF = 1 | Переход если равно |
| Любые | JNE | $a \neq b$ | ZF = 0 | Переход если не равно |
| Со знаком | JL/JNGE | $a < b$ | SF \neq OF | Переход если меньше |
| Со знаком | JLE/JNG | $a \leq b$ | SF \neq OF или ZF = 1 | Переход если меньше или равно |
| Со знаком | JG/JNLE | $a > b$ | SF = OF и ZF = 0 | Переход если больше |
| Со знаком | JGE/JNL | $a \geq b$ | SF = OF | Переход если больше или равно |
| Без знака | JB/JNAE | $a < b$ | CF = 1 | Переход если ниже |

Рис. 3.2: Арифметическое сравнение стр a и b

3.3 Файл листинга и его структура

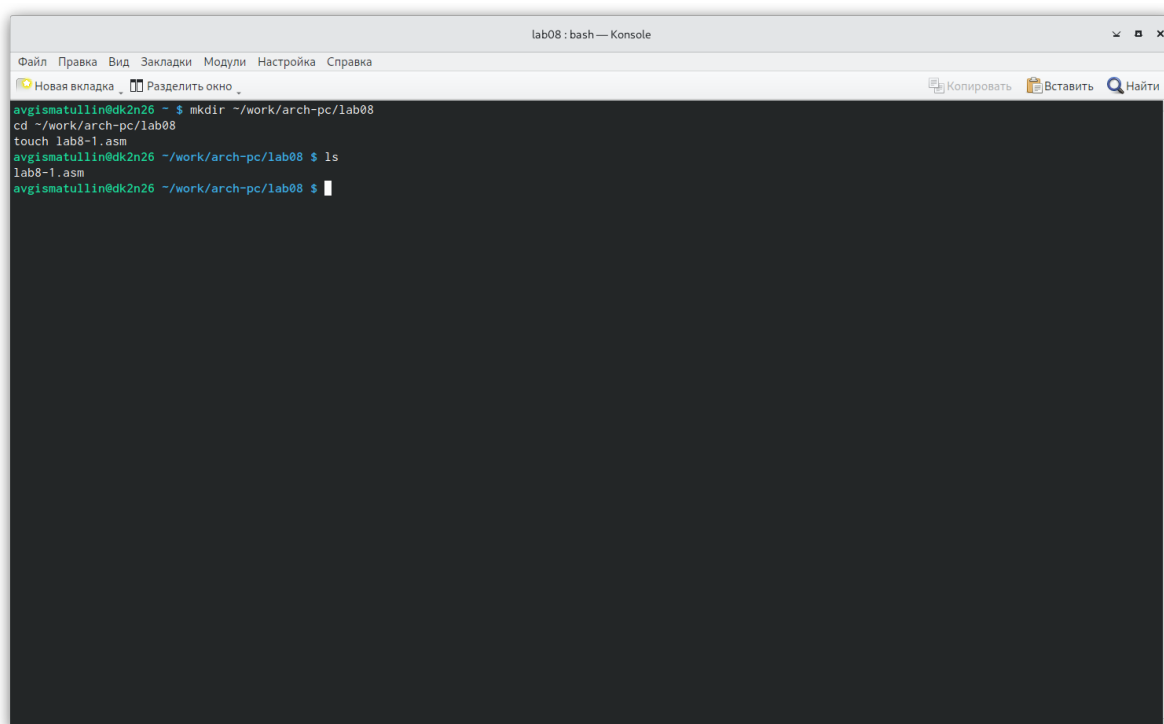
Листинг (в рамках понятийного аппарата NASM) — это один из выходных файлов, создаваемых транслятором. Он имеет текстовый вид и нужен при отладке программы, так как кроме строк самой программы он содержит дополнительную информацию.

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

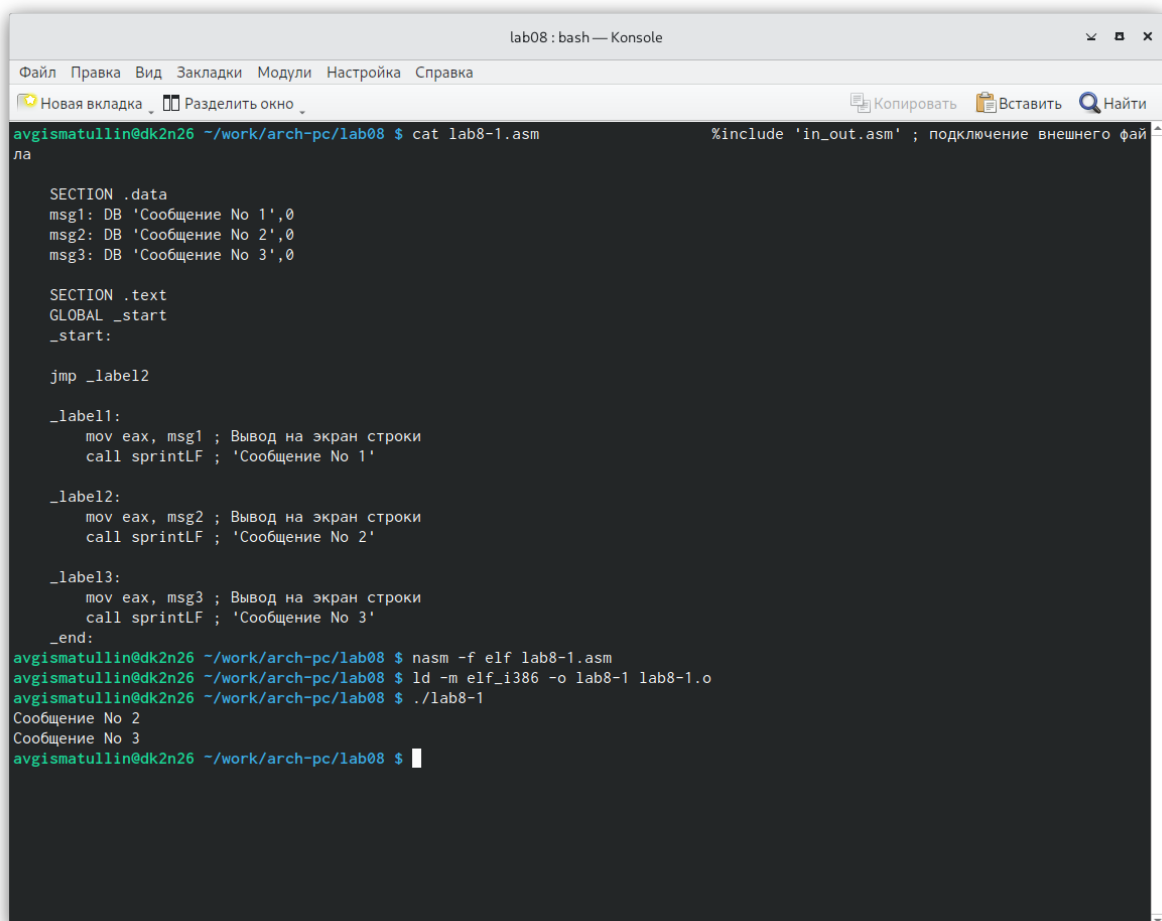
1. Создаем каталог для программ лабораторной работы No 8, перейдем в него и создаем файл lab8-1.asm. (рис. 4.1)



```
lab08: bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно
Копировать  Вставить  Найти
avgismatullin@dk2n26 ~ $ mkdir ~/work/arch-pc/lab08
cd ~/work/arch-pc/lab08
touch lab8-1.asm
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ ls
lab8-1.asm
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $
```

Рис. 4.1: Командная строка. каталог ~/work/arch-pc/lab08

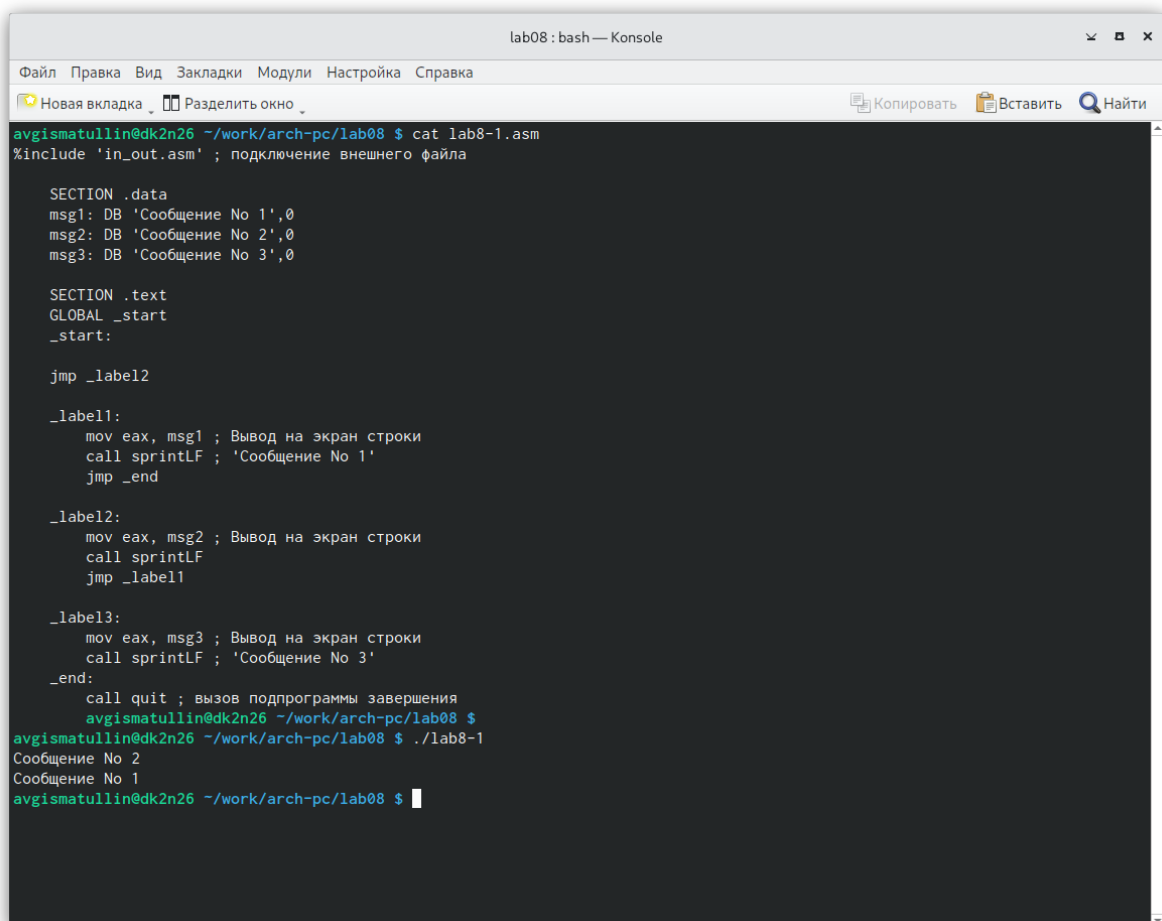
2. Ввели в файл lab8-1.asm текст программы листинга, создали исполняемый файл и посмотрели результат программы. (рис. 4.2)



```
lab08: bash — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно
Копировать Вставить Найти
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ cat lab8-1.asm
%include 'in_out.asm' ; подключение внешнего файла
ла
SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение No 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение No 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение No 3'
_end:
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ ./lab8-1
Сообщение No 2
Сообщение No 3
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $
```

Рис. 4.2: Командная строка. Выполнение программы lab8-1.asm

3. Внесли в файл lab8-1.asm изменение такие, чтобы программа делала другую очередность вывода сообщения, создали исполняемый файл и посмотрели результат программы. (рис. 4.3)



```
lab08 : bash — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно
Копировать Вставить Найти
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ cat lab8-1.asm
%include 'in_out.asm' ; подключение внешнего файла

SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label2

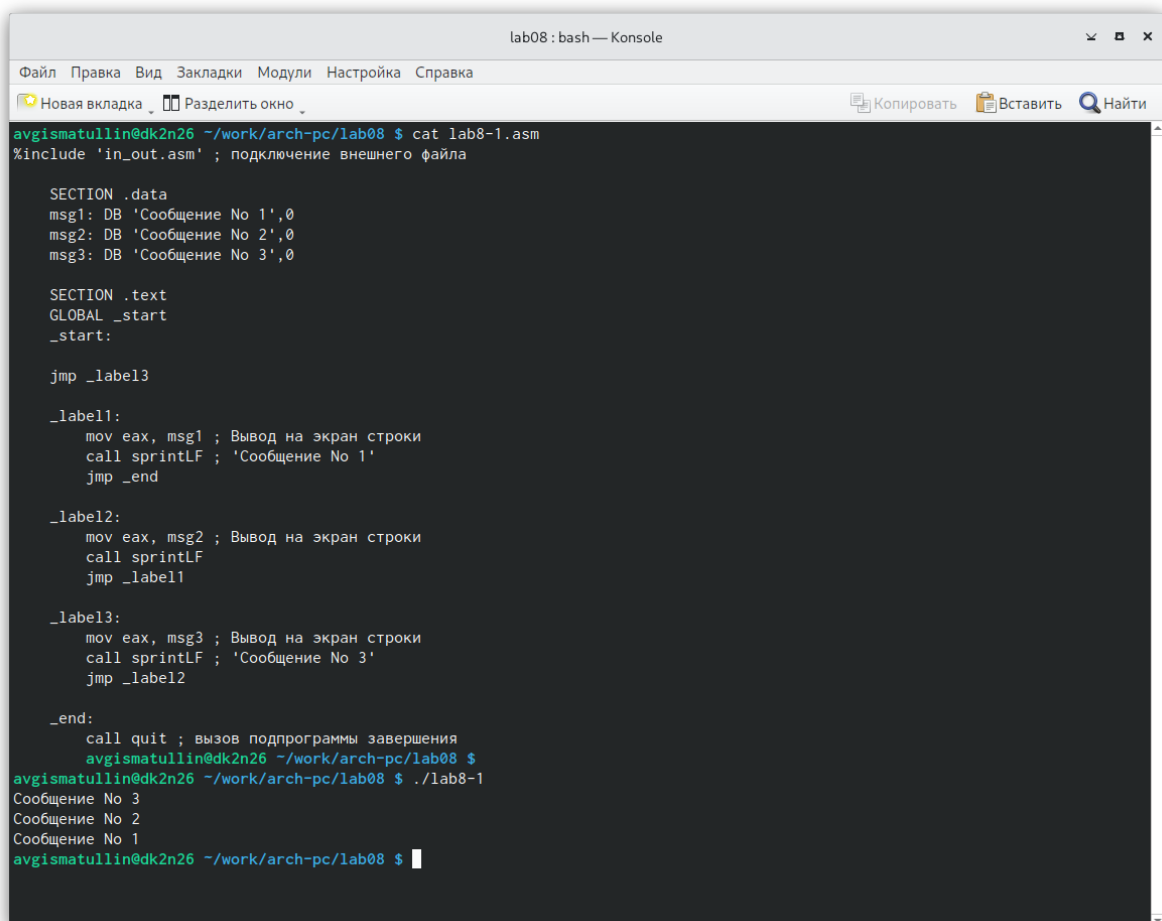
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение No 1'
jmp _end

_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf
jmp _label1

_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение No 3'
_end:
call quit ; вызов подпрограммы завершения
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ ./lab8-1
Сообщение No 2
Сообщение No 1
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $
```

Рис. 4.3: Командная строка. Выполнение измененной программы lab8-1.asm

4. Внесли в файл lab8-1.asm другие изменения - теперь программа выводит сообщения в обратном порядке, создали исполняемый файл и посмотрели результат программы. (рис. 4.4)



```
lab08 : bash — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно
Копировать Вставить Найти

avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ cat lab8-1.asm
%include 'in_out.asm' ; подключение внешнего файла

SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0

SECTION .text
GLOBAL _start
_start:

    jmp _label3

_label1:
    mov eax, msg1 ; Вывод на экран строки
    call sprintf ; 'Сообщение No 1'
    jmp _end

_label2:
    mov eax, msg2 ; Вывод на экран строки
    call sprintf
    jmp _label1

_label3:
    mov eax, msg3 ; Вывод на экран строки
    call sprintf ; 'Сообщение No 3'
    jmp _label2

_end:
    call quit ; вызов подпрограммы завершения
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ ./lab8-1
Сообщение No 3
Сообщение No 2
Сообщение No 1
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $
```

Рис. 4.4: Командная строка. Выполнение измененной программы lab8-1.asm 2

5. Создали файл lab8-2.asm в каталоге ~/work/arch-pc/lab08. Внимательно изучили текст программы из листинга 8.3 и ввели в lab8-2.asm. Проверили работоспособность, введя различные значения В. (рис. 4.5)

```
lab08 : bash — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ ./lab8-2
Введите B: 20
Наибольшее число: 50
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ ./lab8-2
Введите B: 60
Наибольшее число: 60
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ ./lab8-2
Введите B: 50
Наибольшее число: 50
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $
```

Рис. 4.5: Командная строка. Выполнение программы lab8-2.asm 2

6. Получили файл листинга программы lab8-2.asm. Изучили его содержимое.
(рис. 4.6)

```
lab08 : mcedit — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти
lab8-2.lst [----] 0 L:[191+ 0 191/230] *(11596/14815b) 0032 0x020 [*][X]
16 _start:
17 ; ----- Вывод сообщения 'Введите B: '
18 000000E8 B8[00000000] mov eax,msg1
19 000000ED E81DFFFFFF call sprint
20 ; ----- Ввод 'B'
21 000000F2 B9[0A000000] mov ecx,B
22 000000F7 BA0A000000 mov edx,10
23 000000FC E842FFFFFF call sread
24 ; ----- Преобразование 'B' из символа в число
25 00000101 B8[0A000000] mov eax,B
26 00000106 E891FFFFFF call atoi ; Вызов подпрограммы перевода символа в число
27 0000010B A3[0A000000] mov [B],eax ; запись преобразованного числа в 'B'
28 ; ----- Записываем 'A' в переменную 'max'
29 00000110 8B0D[35000000] mov ecx,[A] ; 'ecx = A'
30 00000116 890D[00000000] mov [max],ecx ; 'max = A'
31 ; ----- Сравниваем 'A' и 'C' (как символы)
32 0000011C 3B0D[39000000] cmp ecx,[C] ; Сравниваем 'A' и 'C'
33 00000122 7F0C jg check_B ; если 'A>C', то переход на метку 'check_B',
34 00000124 8B0D[39000000] mov ecx,[C] ; иначе 'ecx = C'
35 0000012A 890D[00000000] mov [max],ecx ; 'max = C'
36 ; ----- Преобразование 'max(A,C)' из символа в число
37 check_B:
38 00000130 B8[00000000] mov eax,max
1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Переместить 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 4.6: Командная строка. Просмотр содержимого файла листинга

Если обратить внимание на строки 18-20, то можно сделать вывод о следующем:

- Исходный текст программы - это просто строка исходной программы с комментариями. Комментарии не генерируют никакого машинного кода, но отдельной строке-комментарию присваивается ее номер (строка 20).
- Строки 18 и 19 содержат так же адрес (смещение машинного кода от начала текущего сегмента, например, 000000E8) и машинный код (B8), который представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности. B8, к примеру, - это инструкция на машинном языке: в регистр eax помещается содержимое msg.

7. Далее мы удаляем один операнд инструкции mov и проверяем файл листинга. (рис. 4.7)

```
lab8-2.lst [----] 90 L:[187+ 7 194/231] *(11890/14901b) 0010 0x00A [*][X]
12
13 section .text
14 global _start
15
16 _start:
17 ; ----- Вывод сообщения 'Введите B: '
18 mov eax
19 ***** error: invalid combination of opcode and operands
20 000000E8 E822FFFFFF call sprint
21 ; ----- Ввод 'B'
22 000000ED B9[0A000000] mov ecx,B
23 000000F2 BA0A000000 mov edx,10
24 000000F7 E847FFFFFF call sread
25 ; ----- Преобразование 'B' из символа в число
26 000000FC B8[0A000000] mov eax,B
27 00000101 E896FFFFFF call atoi ; Вызов подпрограммы перевода символа в число
28 00000106 A3[0A000000] mov [B],eax ; запись преобразованного числа в 'B'
29 ; ----- Записываем 'A' в переменную 'max'
30 0000010B 8B0D[35000000] mov ecx,[A] ; 'ecx = A'
31 00000111 890D[00000000] mov [max],ecx ; 'max = A'
32 ; ----- Сравниваем 'A' и 'C' (как символы)
33 00000117 3B0D[39000000] cmp ecx,[C] ; Сравниваем 'A' и 'C'
34 0000011D 7F0C jg check_B ; если 'A>C', то переход на метку 'check_B'
```

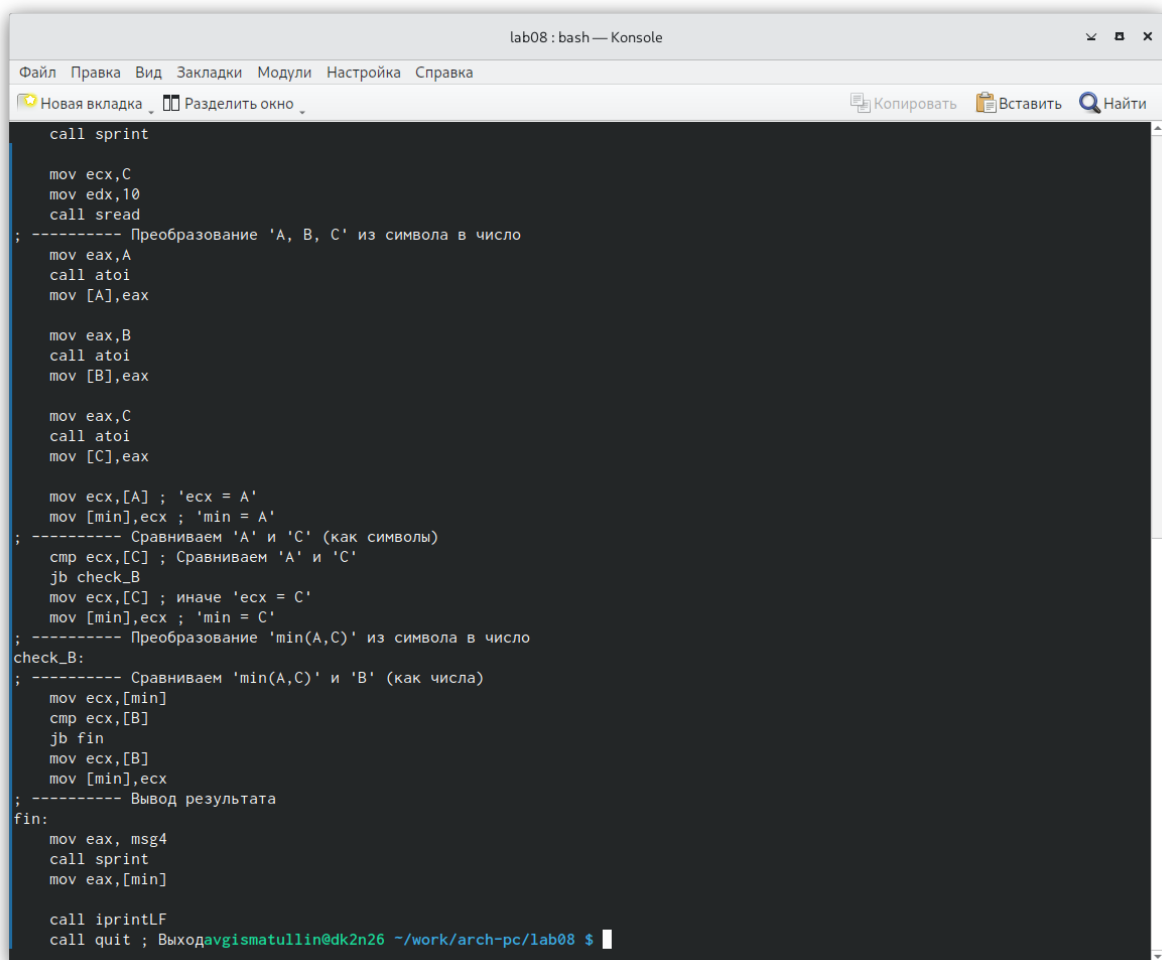
Рис. 4.7: Командная строка. Просмотр содержимого измененного файла листинга

Можем заметить, что на месте ошибки слева отображены звездочки вместо адреса и машинного кода, что уже свидетельствует об ошибке. Кроме того, справа отображено описание самой ошибки.

5 Выполнение задание для самостоятельной работы

Суть задания для самостоятельной работы заключается в том, чтобы написать программу для решения некоторой функции, а также программу, которая будет выводить по трем полученным целочисленным значениям переменных наименьшее из них. Так как на прошлой лабораторной работе я нашел, что мой вариант - 11, то значения я буду брать соответствующие.

1. Выполнение заданий начну с того, что напишу программу (lab8-3.asm), которая будет выводить наименьшее значение из трех. Основываясь на полученных знаниях, я написал ее следующим образом: (рис. 5.1)



```
lab08 : bash — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти

call sprint

mov ecx,C
mov edx,10
call sread
; ----- Преобразование 'A, B, C' из символа в число
mov eax,A
call atoi
mov [A],eax

mov eax,B
call atoi
mov [B],eax

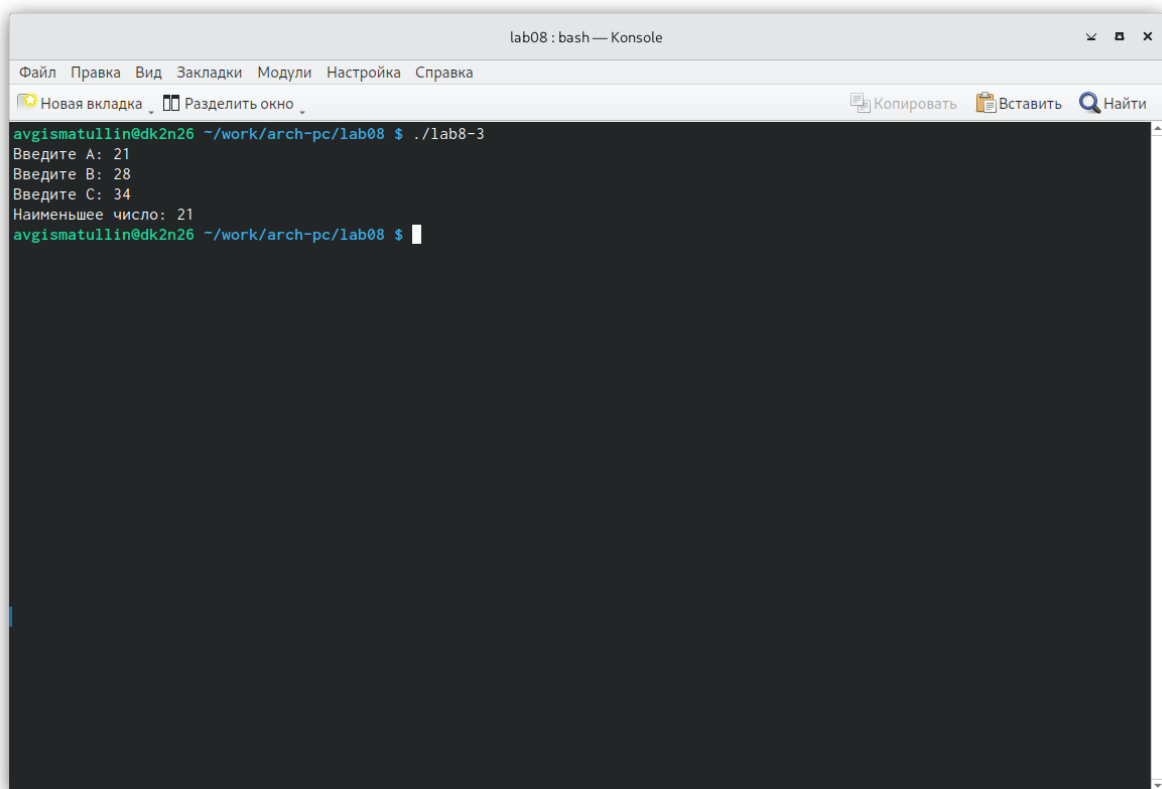
mov eax,C
call atoi
mov [C],eax

mov ecx,[A] ; 'ecx = A'
mov [min],ecx ; 'min = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jb check_B
mov ecx,[C] ; иначе 'ecx = C'
mov [min],ecx ; 'min = C'
; ----- Преобразование 'min(A,C)' из символа в число
check_B:
; ----- Сравниваем 'min(A,C)' и 'B' (как числа)
mov ecx,[min]
cmp ecx,[B]
jb fin
mov ecx,[B]
mov [min],ecx
; ----- Вывод результата
fin:
mov eax,msg4
call sprint
mov eax,[min]

call iprintLF
call quit ; Выходavgismatullin@dk2n26 ~/work/arch-pc/lab08 $
```

Рис. 5.1: Командная строка. Просмотр содержимого файла lab8-3.asm

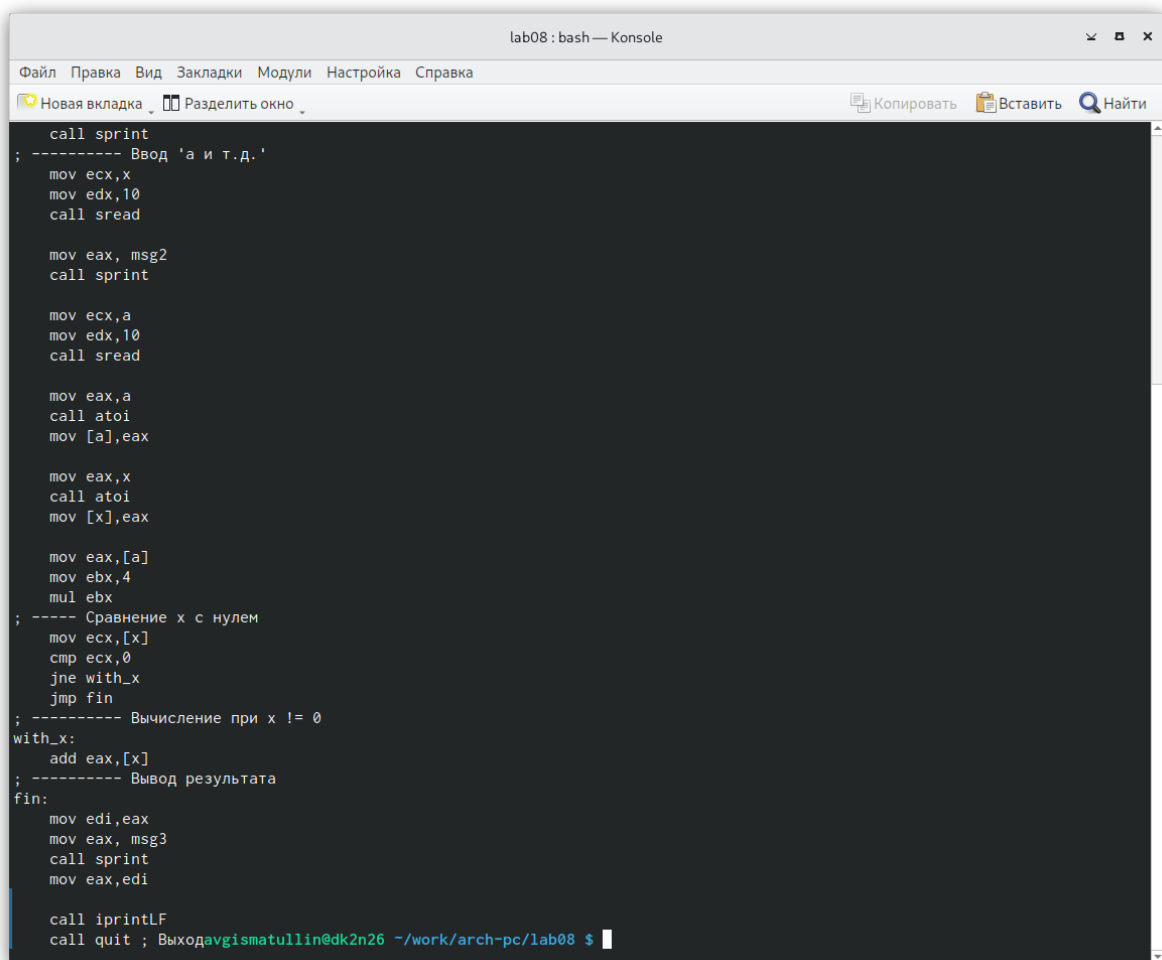
Проверка работоспособности файла: (рис. 5.2)



```
lab08 : bash — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ ./lab8-3
Введите A: 21
Введите B: 28
Введите C: 34
Наименьшее число: 21
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $
```

Рис. 5.2: Командная строка. Выполнение программы lab8-3.asm

2. Далее следует написать код (lab8-4.asm), который по введенным значениям проверяет их на условия и выводит получившийся результат. Условия прописаны в самой функции под вариантом 11. Код выглядит так: (рис. 5.3)

A screenshot of a terminal window titled 'lab08 : bash — Konsole'. The window has a menu bar with 'Файл', 'Правка', 'Вид', 'Закладки', 'Модули', 'Настройка', and 'Справка'. Below the menu bar are icons for 'Новая вкладка', 'Разделить окно', 'Копировать', 'Вставить', and 'Найти'. The main area of the terminal displays assembly code for a program named 'lab8-4.asm'. The code includes comments in Russian and various assembly instructions like 'call sprint', 'mov ecx, x', 'call atoi', and 'call iprintLF'. The code ends with 'call quit ; Выход' and a prompt 'avgismatullin@dk2n26 ~/work/arch-pc/lab08 \$'.

```
call sprint
; ----- Ввод 'а и т.д.'
mov ecx,x
mov edx,10
call sread

mov eax, msg2
call sprint

mov ecx,a
mov edx,10
call sread

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

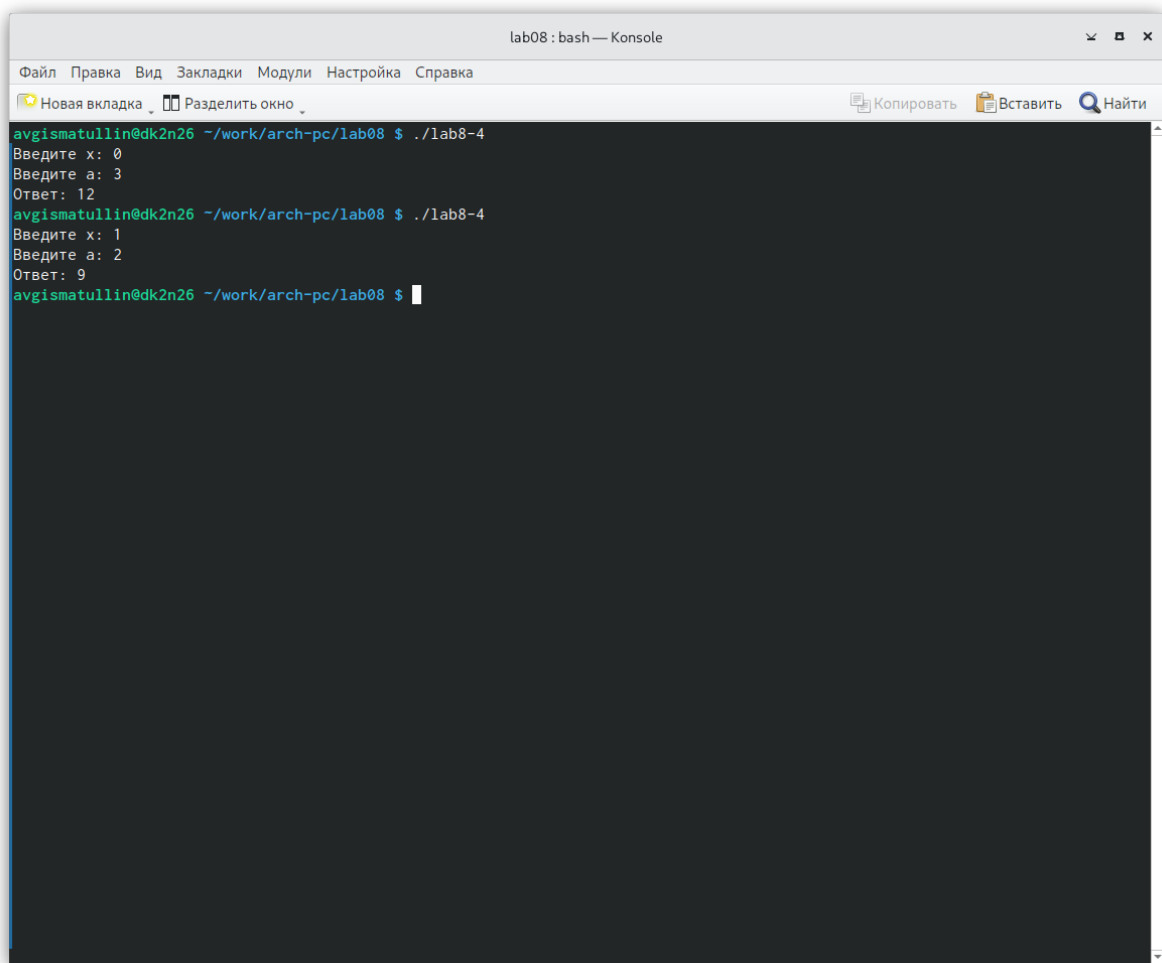
mov eax,x
call atoi
mov [x],eax

mov eax,[a]
mov ebx,4
mul ebx
; ---- Сравнение x с нулем
mov ecx,[x]
cmp ecx,0
jne with_x
jmp fin
; ----- Вычисление при x != 0
with_x:
add eax,[x]
; ----- Вывод результата
fin:
mov edi,eax
mov eax, msg3
call sprint
mov eax,edi

call iprintLF
call quit ; Выходavgismatullin@dk2n26 ~/work/arch-pc/lab08 $
```

Рис. 5.3: Командная строка. Просмотр содержимого файла lab8-4.asm

Проверка работоспособности: (рис. 5.4)



```
lab08 : bash — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ ./lab8-4
Введите x: 0
Введите a: 3
Ответ: 12
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $ ./lab8-4
Введите x: 1
Введите a: 2
Ответ: 9
avgismatullin@dk2n26 ~/work/arch-pc/lab08 $
```

Рис. 5.4: Командная строка. Выполнение программы lab8-4.asm

6 Выводы

В ходе работы я изучил команды условного и безусловного переходов, приобрел навыки написания программ с использованием переходов, а также познакомился с назначением и структурой файла листинга.

Список литературы