Отчёт по лабораторной работе № 7

дисциплина: Архитектура компьютера. Команды безусловного и условного переходов в Nasm. Программирование ветвлений

Студент: Святашова Ксения Евгеньевна

Содержание

# 1 Цель работы

Целью работы является изучение команд условного и безусловного переходов, приобретение навыков написания программ с использованием переходов и знакомство с назначением и структурой файла листинга.

# 2 Теоритическое 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.

Безусловный переход выполняется инструкцией jmp (от англ. jump – прыжок), которая включает в себя адрес перехода, куда следует передать управление. 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.

Для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра 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.

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.

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.

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).

# 3 Выполнение лабораторной работы

## 3.1 Реализация переходов в NASM

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

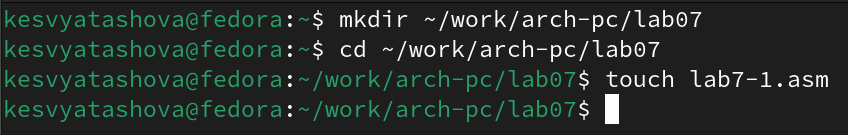


Рис. 1: Каталог lab07

1. Инструкция jmp в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции jmp. Введем в файл lab7-1.asm текст программы из листинга 7.1.(рис. 2):

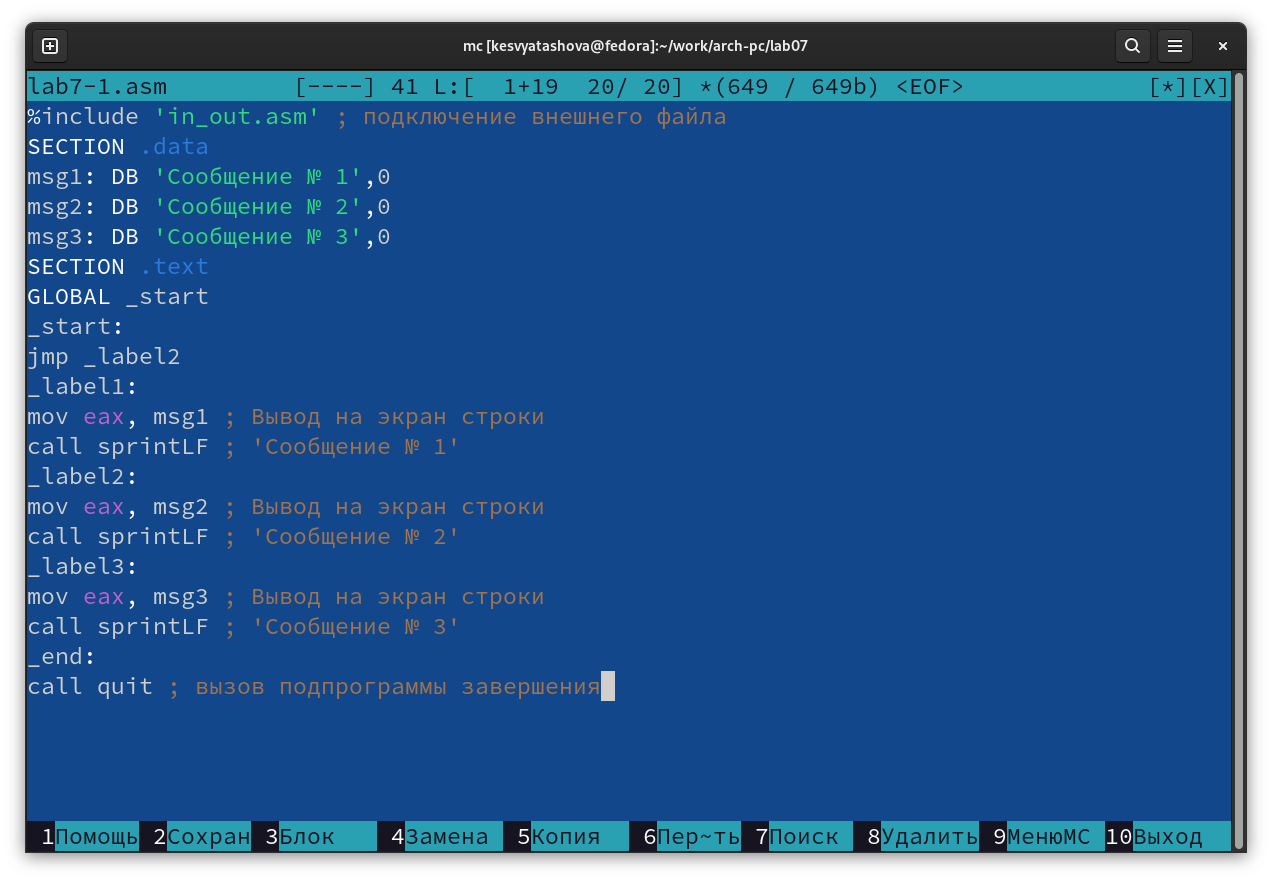


Рис. 2: Ввод текста из листинга 7.1

**Листинг 7.1. Программа с использованием инструкции jmp**

%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 sprintLF ; ‘Сообщение № 1’

\_label2:

mov eax, msg2 ; Вывод на экран строки

call sprintLF ; ‘Сообщение № 2’

\_label3:

mov eax, msg3 ; Вывод на экран строки

call sprintLF ; ‘Сообщение № 3’

\_end:

call quit ; вызов подпрограммы завершения

Создадим исполняемый файл и запустим его(рис. 3):

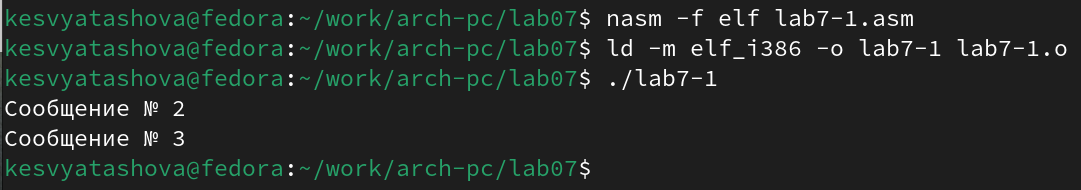


Рис. 3: Запуск исполняемого файла

Таким образом, использование инструкции jmp \_label2 меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки \_label2, пропустив вывод первого сообщения.

Изменим программу так, 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).

Изменим текст программы в соответствии с листингом 7.2.(рис. 4):

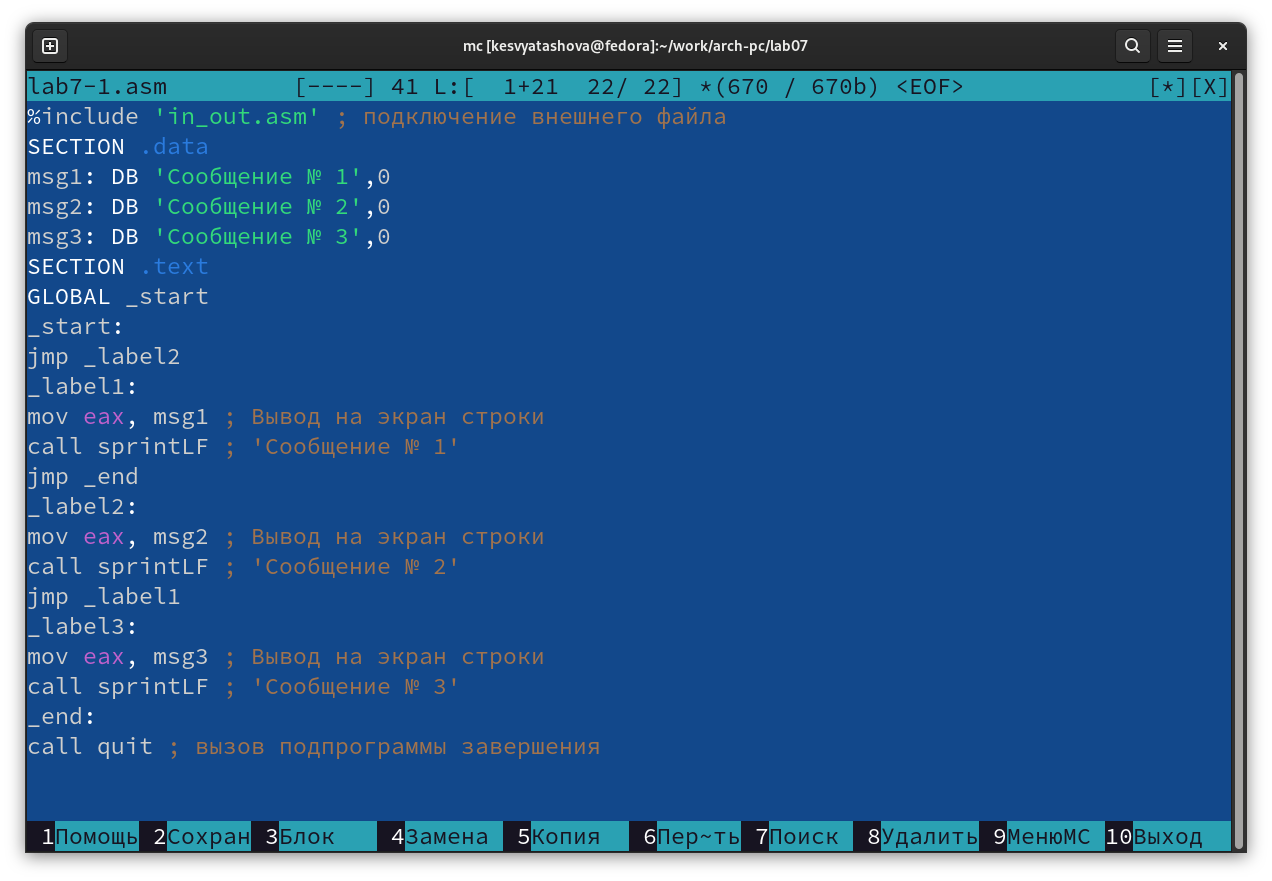


Рис. 4: Ввод текста из листинга 7.2

**Листинг 7.2. Программа с использованием инструкции jmp**

%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 sprintLF ; ‘Сообщение № 1’

jmp \_end

label2:

mov eax, msg2 ; Вывод на экран строки

call sprintLF ; ‘Сообщение № 2’

jmp \_label1

\_label3:

mov eax, msg3 ; Вывод на экран строки

call sprintLF ; ‘Сообщение № 3’

\_end:

call quit ; вызов подпрограммы завершения

Создадим исполняемый файл и запустим его(рис. 5):

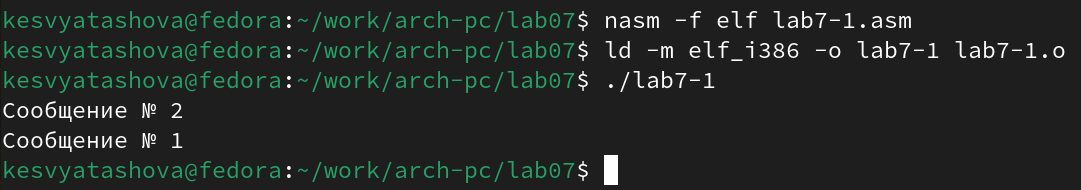


Рис. 5: Запуск исполняемого файла

Изменим текст программы(рис. 6) добавив или изменив инструкции jmp, чтобы вывод программы был следующим(рис. 7):

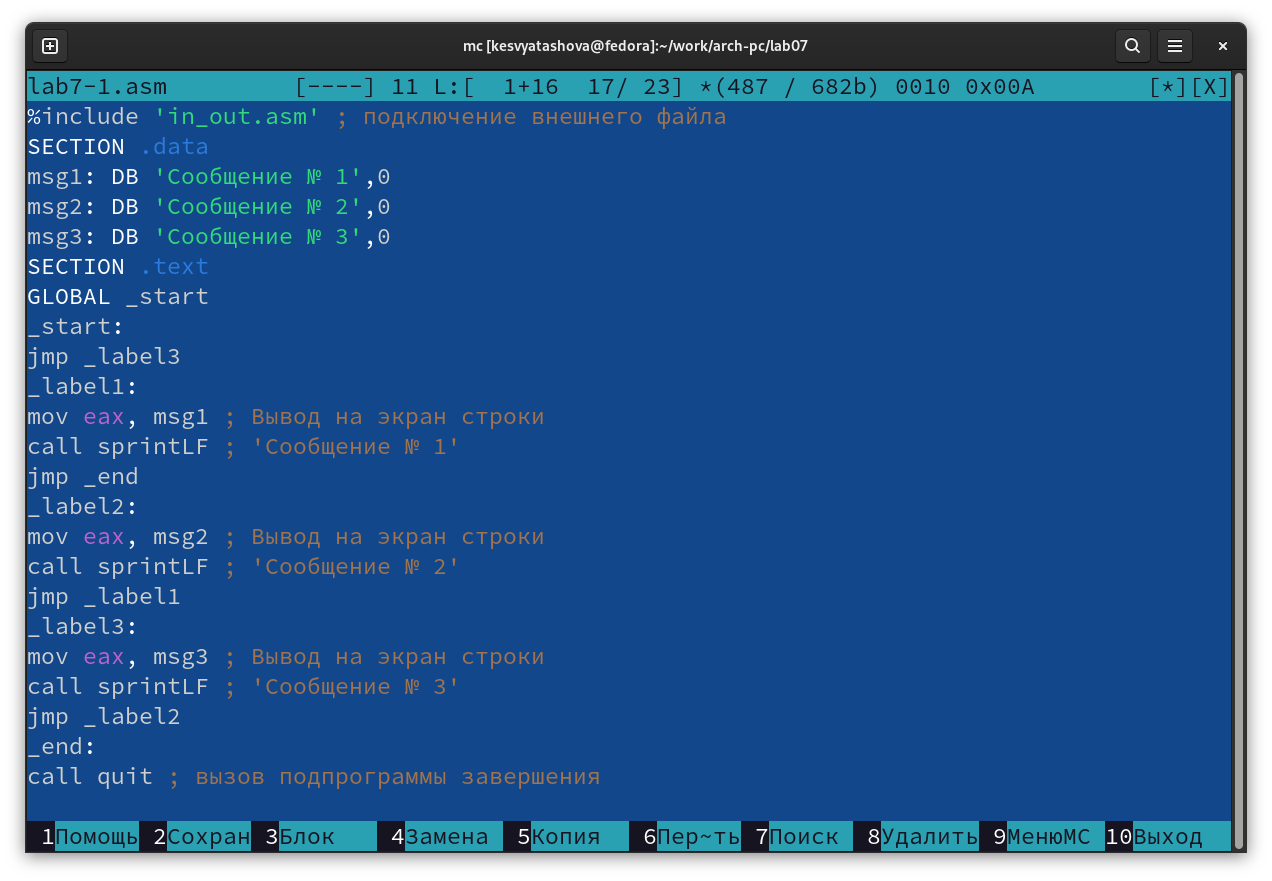


Рис. 6: Изменение текста программы

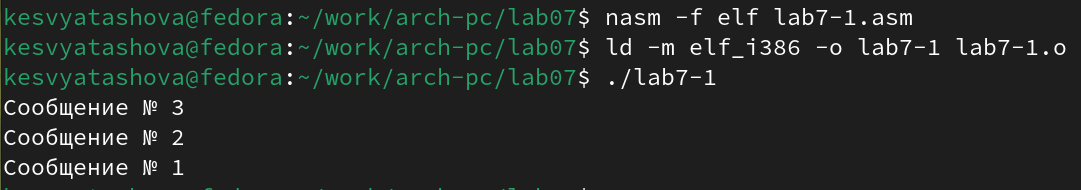


Рис. 7: Запуск исполняемого файла

1. . 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(рис. 8):



Рис. 8: Создание файла lab7-2.asm

Далее в этот файл введем текст программы из листинга 7.3(рис. 9):

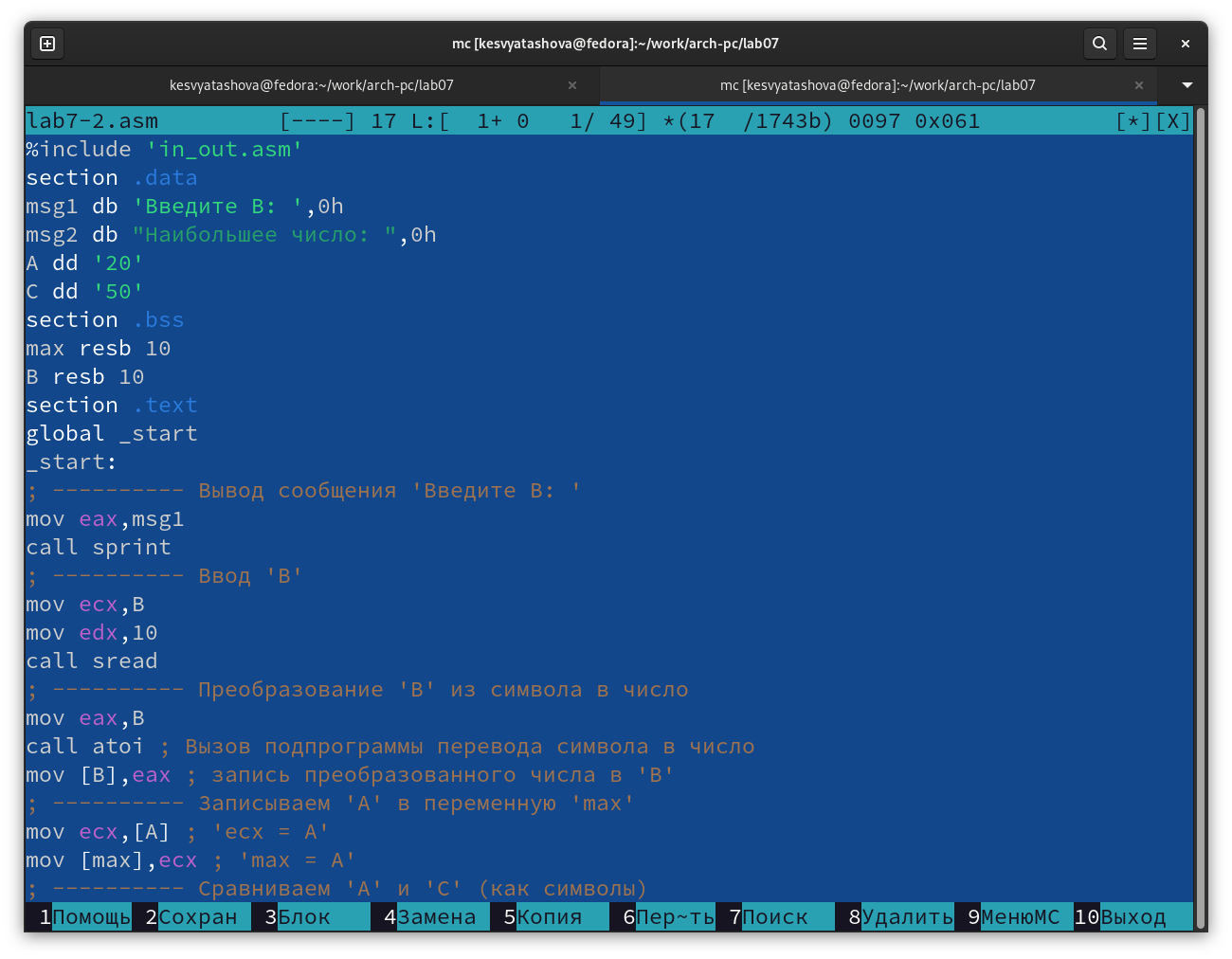


Рис. 9: Ввод текста из листинга 7.3

**Листинг 7.3. Программа, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: A,B и C.**

%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 ‘С’ (как символы)

cmp ecx,[C] ; Сравниваем ‘A’ и ‘С’

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

call sprint ; Вывод сообщения ‘Наибольшее число:’

mov eax,[max]

call iprintLF ; Вывод ‘max(A,B,C)’

call quit ; Выход

Создадим исполняемый файл и запустим его для разных значений B (рис. 10) и (рис. 11):

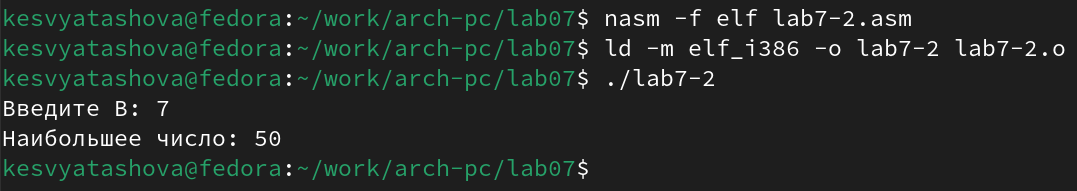


Рис. 10: Запуск исполняемого файла

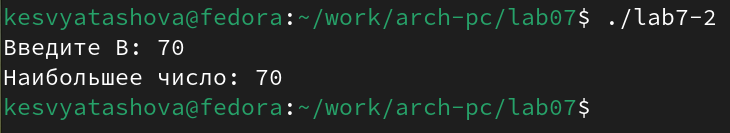


Рис. 11: Запуск исполняемого файла

## 3.2 Изучение структуры файлы листинга

1. Обычно nasm создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ -l и задав имя файла листинга в командной строке.

Создадим файл листинга для программы из файла lab7-2.asm(рис. 12) и откроем этот файл с помощью текстового редактора mcedit(рис. 13):

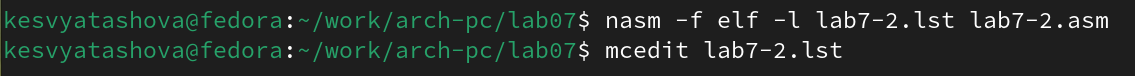


Рис. 12: Создание файла листинга lab7-2.asm

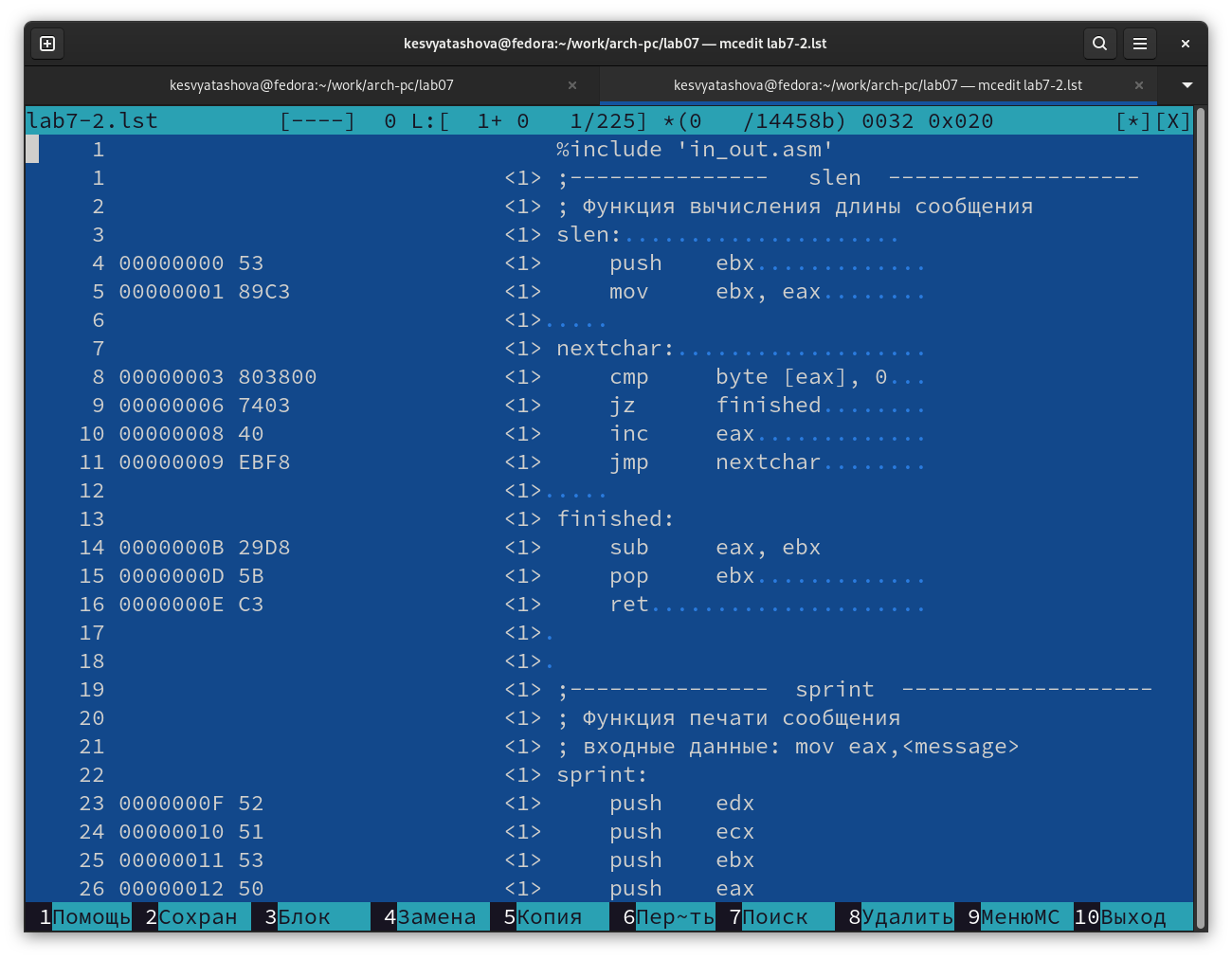


Рис. 13: Файл листинга

Внимательно ознакомившись с форматом и содержимым файла листинга, можно объяснить содержимое его строк. Например:

1. Это строка(рис. 14) находится находится на 22 месте, ее адрес “00000106”, машинный код - “E891FFFFFF”. call atoi означает, вызов подпрограммы перевода символа, лежащего в строке выше, в число.

Рис. 14: Строка листинга

Рис. 14: Строка листинга

1. Это строка(рис. 15) находится находится на 25 месте, ее адрес “00000110”, машинный код - “8BOD[35000000]”. mov ecx,[A] означает, что в регистр ecx мы записываем число, хранящееся в переменной A.

Рис. 15: Строка листинга

Рис. 15: Строка листинга

1. Это строка(рис. 16) находится находится на 36 месте, ее адрес “00000135”, машинный код - “A3[00000000]”. mov [max],eax означает, запись преобразованного числа из регистра eax в max.

Рис. 16: Строка листинга

Рис. 16: Строка листинга

В строке mov eax,max я удалила операнду max и попробывала выполните трансляцию с получением файла листинга(рис. 17):

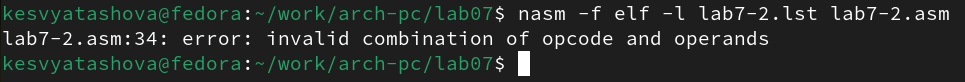


Рис. 17: Попытка трансляции файла

При трансляции файла выходит ошибка, потому что для корректной работы программы нужно два операнда. В файле листинга показывается, где совершена ошибка и почему она выходит(рис. 18):

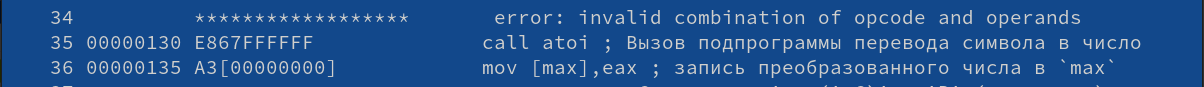


Рис. 18: Ошибка в файле листинга

# 4 Задания для самостоятельной работы

1. Создадим файл lab7-3.asm для выполнения первого задания самостоятельной работы(рис. 19):

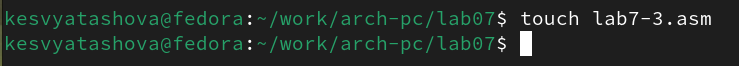


Рис. 19: Создание файла lab7-3.asm

Напишем программу(рис. 20) нахождения наименьшей из 3 целочисленных переменных a,b и c. Значения переменных я выбрала в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Мой вариант - 9.



Рис. 20: Программа вычисления выражение из варианта 9

Создадим исполняемый файл и проверим его работу(рис. 21)

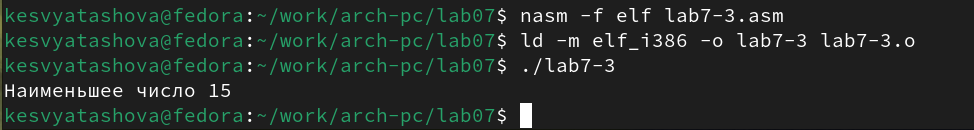


Рис. 21: Запуск исполняемого файла

1. Создадим файл lab7-4.asm для выполнения второго задания самостоятельной работы(рис. 22):

Рис. 22: Создание файла lab7-4.asm

Рис. 22: Создание файла lab7-4.asm

Напишем программу(рис. 23), которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции f(x) и выводит результат вычислений. Вид функции f(x) я выбрала также в соответствии с вариантом, полученным при выполнении лабораторной работы № 6.

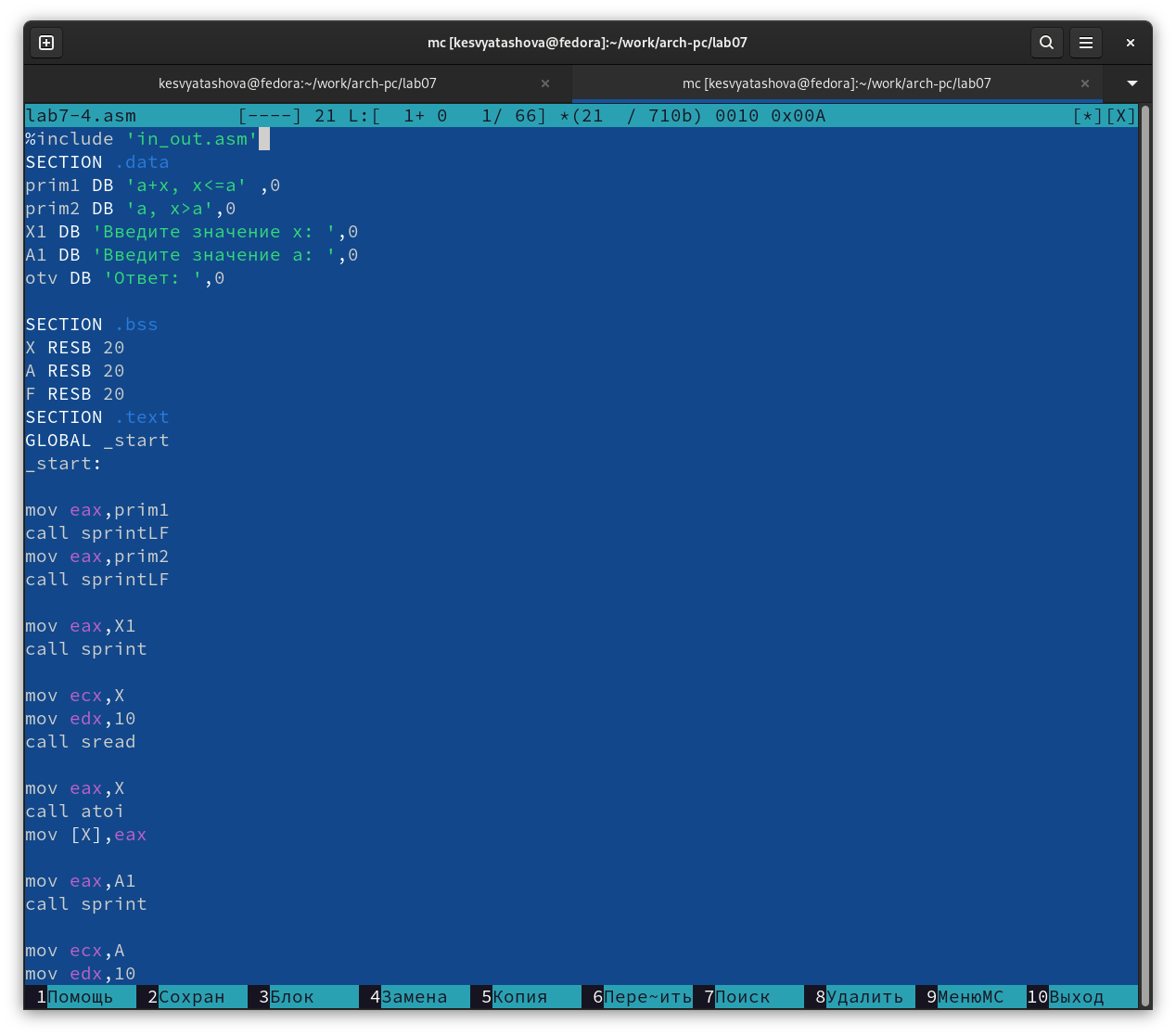


Рис. 23: Программа вычисления выражение из варианта 9

Создадим исполняемый файл и проверьте его работу для значений x и a(рис. 24) и (рис. 25):

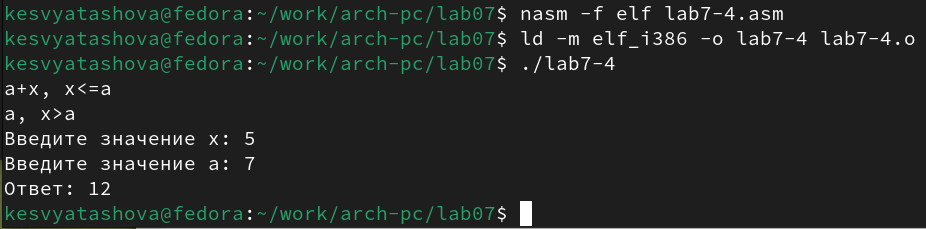


Рис. 24: Запуск исполняемого файла

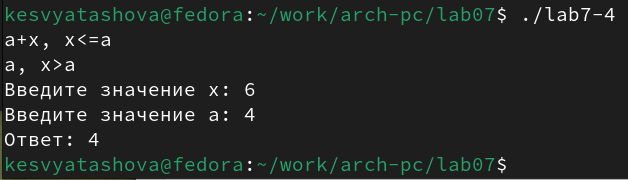


Рис. 25: Запуск исполняемого файла

Текст программы:

%include ‘in\_out.asm’

SECTION .data

prim1 DB ‘a+x, x<=a’ ,0

prim2 DB ‘a, x>a’,0

X1 DB ‘Введите значение x:’,0

A1 DB ‘Введите значение a:’,0

otv DB ‘Ответ:’,0

SECTION .bss

X RESB 20

A RESB 20

F RESB 20

SECTION .text

GLOBAL \_start

\_start:

mov eax,prim1

call sprintLF

mov eax,prim2

call sprintLF

mov eax,X1

call sprint

mov ecx,X

mov edx,10

call sread

mov eax,X

call atoi

mov [X],eax

mov eax,A1

call sprint

mov ecx,A

mov edx,10

call sread

mov eax,A

call atoi

mov [A],eax

mov ecx,[X]

mov [F],ecx

cmp [A],ecx

jg check\_or

mov eax,[A]

mov [F],eax

jmp fin

check\_or:

mov eax,[X]

add eax,[A]

mov [F],eax

fin:

mov eax,otv

call sprint

mov eax,[F]

call iprintLF call quit

# 5 Вывод

В результате выполнения работы я изучила команды условного и безусловного переходов, приобрела навыки написания программ с использованием переходов и познакомилась с назначением и структурой файла листинга.