Отчёт по лабораторной работе №7

дисциплина: Архитектура компьютера

Аносов Даниил Игоревич

Содержание

# 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. Напишите программу нахождения наименьшей из 3 целочисленных переменных и . Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Создайте исполняемый файл и проверьте его работу.
2. Напишите программу, которая для введенных с клавиатуры значений и вычисляет значение заданной функции и выводит результат вычислений. Вид функции выбрать из таблицы 7.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Создайте исполняемый файл и проверьте его работу для значений и из 7.6.

# 3 Выполнение лабораторной работы

## 3.1 Реализация переходов в NASM

Откроем терминал и создадим каталог для программ лабораторной работы №7. В новом каталоге создадим файл для первой программы *lab7-1.asm*. (рис. 1).

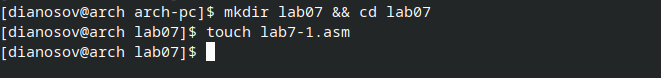


Рис. 1: Создание каталога для программ

Введём в этот файл текст программы из предложенного листинга. (рис. 2).

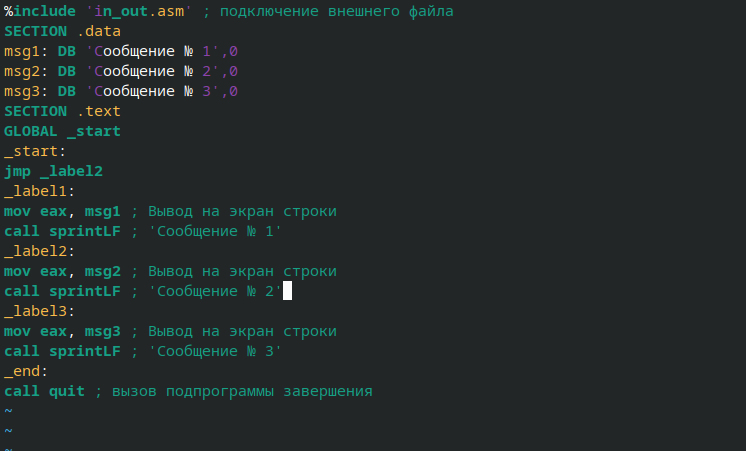


Рис. 2: Открытый Vim

Скомпилируем и запустим программу, предварительно скопировав из каталога предыдущей лабораторной работы вспомогательный файл с подпрограммами *in\_out.asm* (рис. 3).

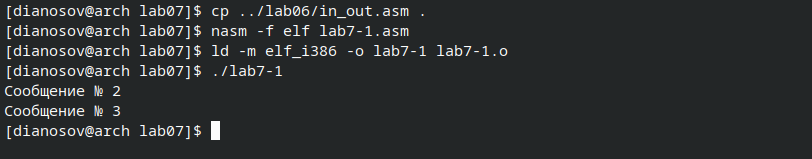


Рис. 3: Компиляция и первый запуск программы

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).

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

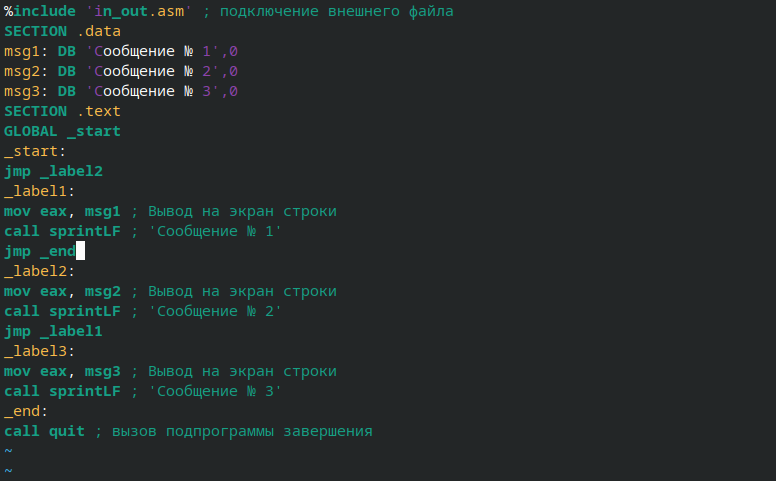


Рис. 4: Vim с обновленной программой

Скомпилируем и запустим программу. (рис. 5).

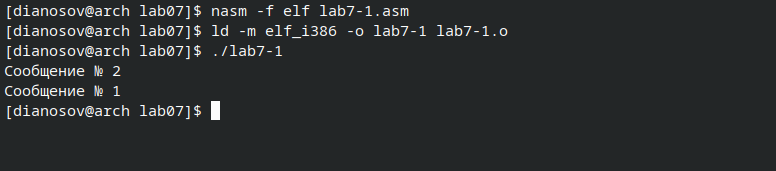


Рис. 5: Повторная компиляция и запуск программы

Теперь изменим программу так, чтобы сообщения выводились в порядке 3,2,1. (рис. 6).

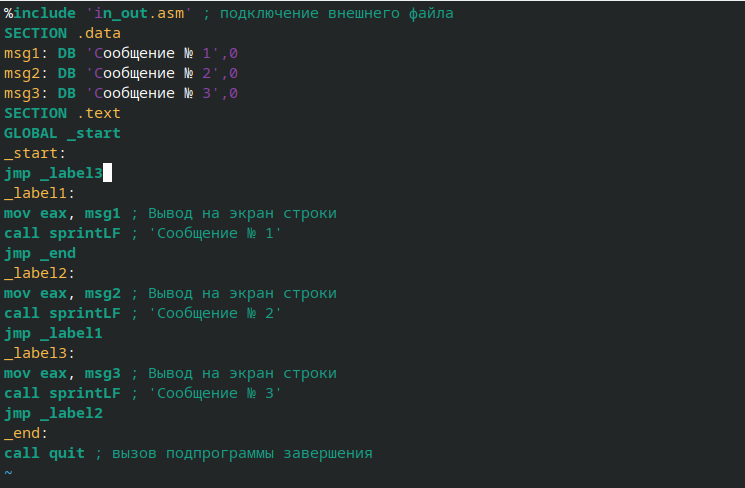


Рис. 6: Vim с обновленной программой

Сохраним, скомпилируем и запустим новую программу. (рис. 7).

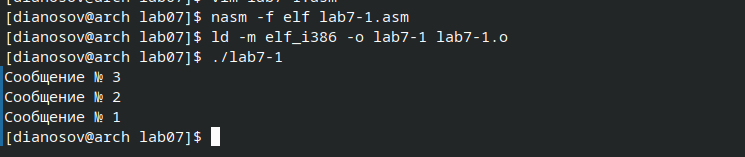


Рис. 7: Компиляция и запуск новой программы

Как видно, программа работает корректно.

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 вводится с клавиатуры.

Текст новой программы, взятый из листинга, введем в новый файл *lab7-2.asm*



Рис. 8: Редактирование файла *lab7-2.asm*

Создадим исполняемый файл и проверим работу программы. (рис. 9).

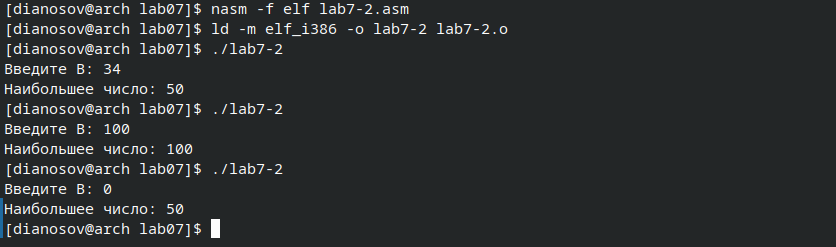


Рис. 9: Повторная компиляция и запуск новой программы

Обратим внимание, в данном примере переменные A и С сравниваются как символы, а переменная B и максимум из A и С как числа (для этого используется функция atoi преобразования символа в число). Это сделано для демонстрации того, как сравниваются данные. Данную программу можно упростить и сравнивать все 3 переменные как символы (т.е. не использовать функцию atoi). Однако если переменные преобразовать из символов числа, над ними можно корректно проводить арифметические операции.

## 3.2 Изучение структуры файлы листинга

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. 10). Будем использовать команду:

nasm -f elf -l lab7-2.lst lab7-2.asm

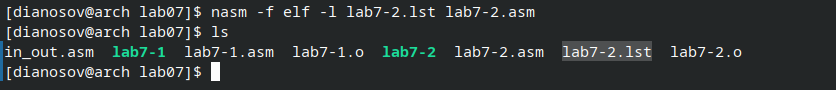


Рис. 10: Создание файла листинга для *lab7-2.asm*

Откроем файл листинга в редакторе **Vim** и обратим внимание на строки 45-47. (рис. 11).

45 00000159 B8[13000000] mov eax, msg2  
 46 0000015E E8ACFEFFFF call sprint ; Вывод сообщения 'Наибольшее число: '  
 47 00000163 A1[00000000] mov eax,[max]

Машинные коды в строках 45 и 47 имеют в конце значения в квадратных скобках, так как они соответствуют вызовам инструкций с двумя операндами. Строка 46 же отвечает вызову функции sprint, поэтому её машинный код (E8ACFEFFFF) - это единая строка. Итого, машинный код, отвечающий инструкции с одним операндом - это слитная строка, а если операндов больше, чем один - то машинный код будет содержать часть в квадратных скобках.

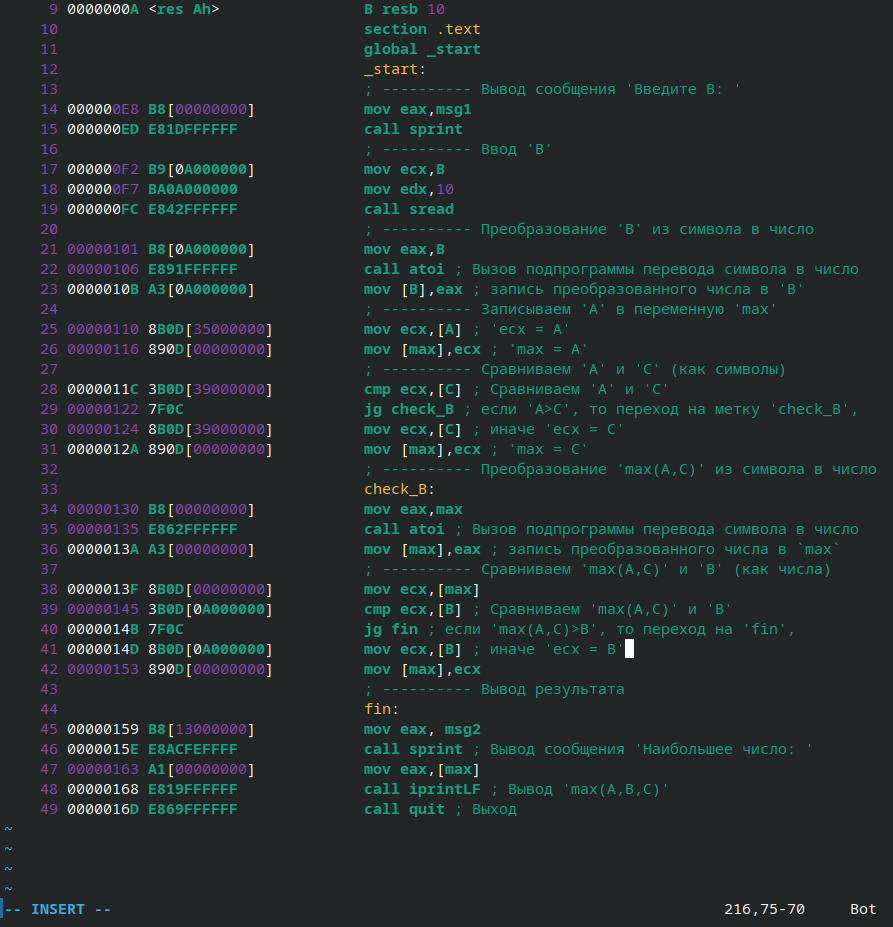


Рис. 11: Содержимое файла листинга

# 4 Задание для самостоятельной работы

## 4.1 Нахождение наименьшего из трёх чисел

### 4.1.1 Введение

Напишем программу, которая находит наименьшее из трёх введённых пользователем чисел. Для этого воспользуемся кодом программы из предыдущих примеров, и изменим его должным образом. Скопируем файл *lab7-2.asm* и дадим новой программе имя *task1.asm*. (рис. 12, 13).

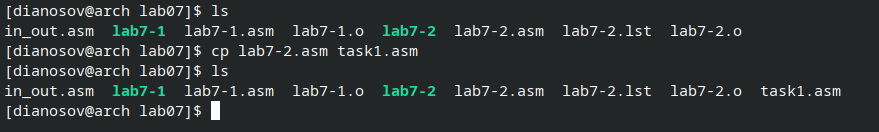


Рис. 12: Копирование файла *lab7-2.asm*

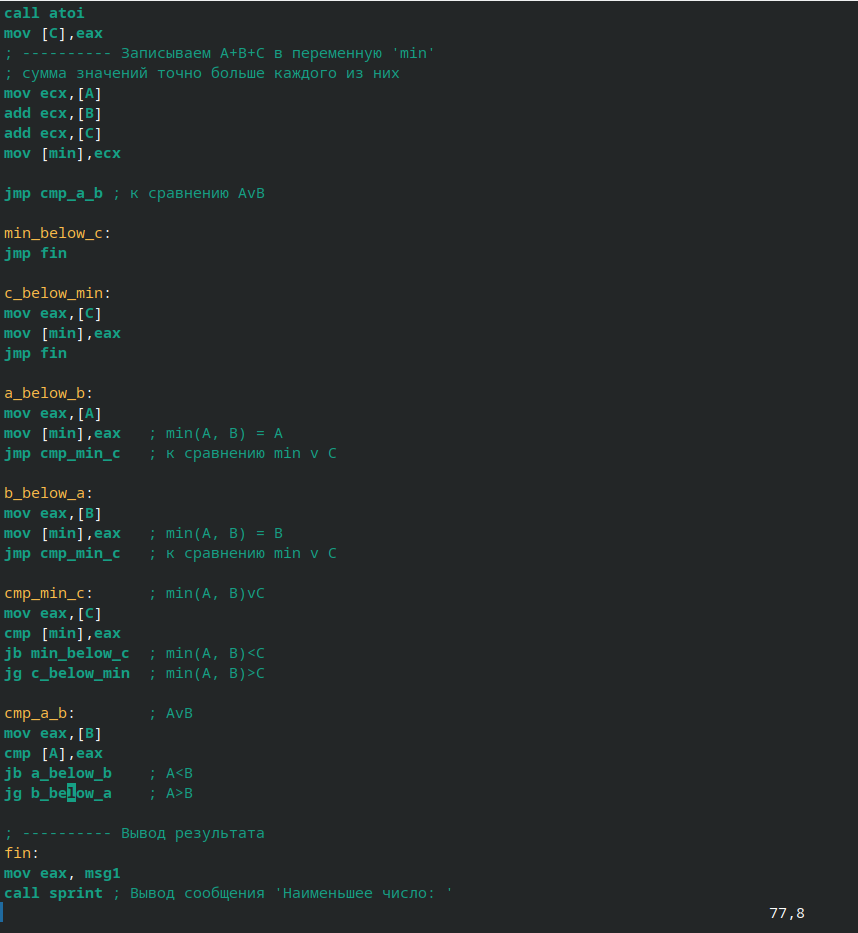


Рис. 13: Редактирование файла *task1.asm*

### 4.1.2 Алгоритм работы программы. Тестирование

Алгоритм работы программы: сравниваем A и B, минимальное из них кладём в переменную min. Сравниваем min с C. Минимальное из них - минимальное из всех чисел - результат работы программы. Пример на рис. 14. Скомпилируем и запустим новую программу.

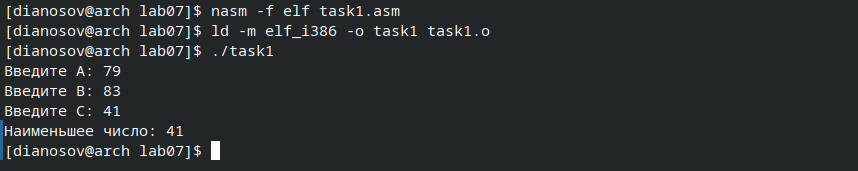


Рис. 14: Компиляция и запуск *task1.asm*

Тестовый запуск был проведён со значениями 79, 83, 41, соответствующими варианту №6, полученному в предыдущей лабораторной работе.

### 4.1.3 Исходный код программы

Исходный текст *task1.asm* приведён ниже.

%include 'in\_out.asm'  
section .data  
  
msga db 'Введите A: ',0h  
msgb db 'Введите B: ',0h  
msgc db 'Введите C: ',0h  
msg1 db "Наименьшее число: ",0h  
section .bss  
min resb 10  
A resb 10 ; резервируем место не  
B resb 10 ; только под B,  
C resb 10 ; но и под A и C  
section .text  
global \_start  
\_start:  
; ---------- Ввод A, B, C  
mov eax,msga  
call sprint  
mov ecx,A  
mov edx, 10  
call sread  
mov eax,msgb  
call sprint  
mov ecx,B  
call sread  
mov eax,msgc  
call sprint  
mov ecx,C  
call sread  
; ---------- Преобразование A,B,C из символов в числа  
mov eax,A  
call atoi ; Вызов подпрограммы перевода символа в число  
mov [A],eax ; запись преобразованного числа в 'A'  
mov eax,B  
call atoi  
mov [B],eax  
mov eax,C  
call atoi  
mov [C],eax  
; ---------- Записываем A+B+C в переменную 'min'  
; сумма значений точно больше каждого из них  
mov ecx,[A]  
add ecx,[B]  
add ecx,[C]  
mov [min],ecx  
  
jmp cmp\_a\_b ; к сравнению AvB  
  
min\_below\_c:  
jmp fin  
  
c\_below\_min:  
mov eax,[C]  
mov [min],eax  
jmp fin  
  
a\_below\_b:  
mov eax,[A]  
mov [min],eax ; min(A, B) = A  
jmp cmp\_min\_c ; к сравнению min v C  
  
b\_below\_a:  
mov eax,[B]  
mov [min],eax ; min(A, B) = B  
jmp cmp\_min\_c ; к сравнению min v C  
  
cmp\_min\_c: ; min(A, B)vC  
mov eax,[C]  
cmp [min],eax  
jb min\_below\_c ; min(A, B)<C  
jg c\_below\_min ; min(A, B)>C  
  
cmp\_a\_b: ; AvB  
mov eax,[B]  
cmp [A],eax  
jb a\_below\_b ; A<B  
jg b\_below\_a ; A>B  
  
; ---------- Вывод результата  
fin:  
mov eax, msg1  
call sprint ; Вывод сообщения 'Наименьшее число: '  
mov eax,[min]  
call iprintLF ; Вывод 'min(A,B,C)'  
call quit ; Выход

## 4.2 Вычисление значения функции

### 4.2.1 Введение

Требуется написать программу, вычисляющую значение функции в зависимости от значений . Я получил вариант №6 в процессе выполнений заданий предыдущей лабораторной работы. Этому варианту соответствует следующая функция:

Для написания программы создадим файл *task2.asm*, откроем его в редакторе **Vim**. (рис. 15).

Рис. 15: Создание файла task2.asm

Рис. 15: Создание файла *task2.asm*

### 4.2.2 Алгоритм работы программы. Тестирование

Программа работает по следующей логике: Сравниваем значения и . Если , то переходим к метке a\_equals\_x. В этой части программы приравниваем результат к и прибавляем к нему, а затем переходим к метке завершения fin. Иначе, если значения не равны, переходим к метке a\_nequals\_x: приравниваем результат к , умножаем его на и переходим к завершению (fin). Под меткой fin выводим сообщение с результатом и вызываем сигнал прерывания.

Фрагмент когда приведён на рис. 16.



Рис. 16: **Vim** с файлом *task2.asm*

Проверим работу программы для входных данных (рис. 17).

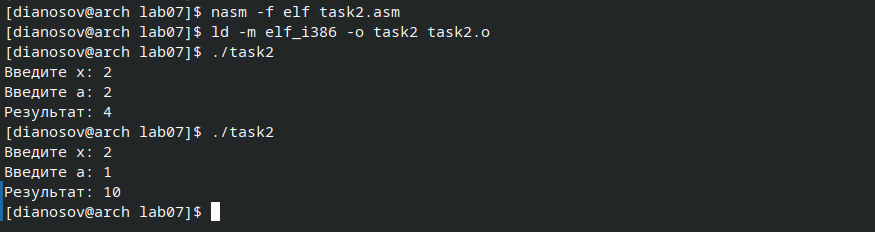


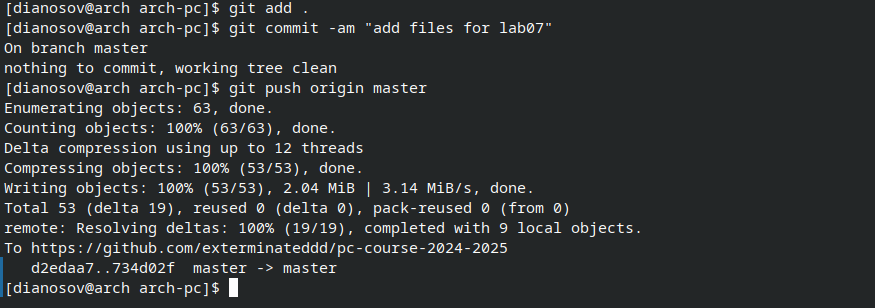
Рис. 17: Тестирование программы *task2.asm*

Как видим, программа работает корректно.

### 4.2.3 Исходный код программы

Исходный код программы *task2.asm* приведён ниже.

%include 'in\_out.asm'  
section .data  
msgx db 'Введите x: ',0h  
msga db 'Введите a: ',0h  
msg1 db "Результат: ",0h  
  
section .bss  
x resb 10 ; резервируем место под x  
a resb 10 ; под а  
res resb 10 ; и под результат  
  
section .text  
global \_start  
\_start:  
; ------ Ввод значений х, а  
mov eax,msgx  
call sprint  
mov ecx,x  
mov edx,10  
call sread  
mov eax,msga  
call sprint  
mov ecx,a  
call sread  
; ------ Преобразование х, а в числа  
mov eax,x  
call atoi ; вызов подпрограммы перевода символа в число  
mov [x],eax ; запись преобразованного числа в 'x'  
mov eax,a  
call atoi  
mov [a],eax  
; ------ Логическая часть программы  
mov eax, [a]  
mov ebx, [x]  
cmp ebx, eax ; сравнение а и х  
je a\_equals\_x  
jne a\_nequals\_x  
  
a\_equals\_x:  
mov ecx, [x]  
mov [res], ecx ; ecx := x  
mov ecx, [a]  
add [res], ecx ; res := res + a = x + a  
jmp fin  
  
a\_nequals\_x:  
mov ecx, [x]  
mov [res], ecx ; res := x  
mov ecx, 5  
mov eax, [res] ; eax := res  
mul ecx ; eax := res \* 5 = 5x  
mov [res], eax ; res := eax = 5x  
jmp fin  
  
fin:  
mov eax, msg1  
call sprint ; Вывод сообщения 'Результат: '  
mov eax,[res]  
call iprintLF ; Вывод res  
call quit ; Выход

Задание выполнено, загрузим новую версию проекта курса на GitHub. 

# 5 Выводы

В ходе выполнения лабораторной работы были изучены команды условного и безусловного переходов. Приобретены навыки написания программ с использованием переходов.