Лабораторнрая работа №7

Команды безусловного и условного переходов в Nasm. Программирование ветвлений.

Соловьев Богдан Михайлович НКАбд-05-23

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 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.

Безусловный переход выполняется инструкцией jmp (от англ. jump – прыжок), 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

Команда условного перехода имеет вид

j label

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

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 jnbe). Программист выбирает, какую из них применить, чтобы получить более простой для понимания текст программы.

# 3 Выполнение лабораторной работы

Создаю файл lab7-1.asm (рис. [1](#fig:001)).

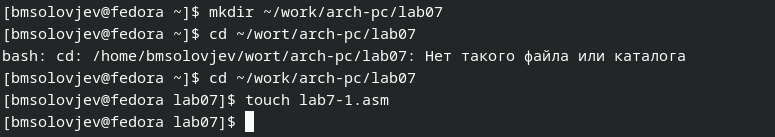


Figure 1: Создание файла

Ввожу в файл lab7-1.asm код программы с использованием jmp (рис. [2](#fig:002)).

%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

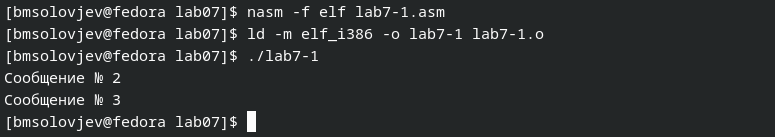


Figure 2: Работа программы с функцией jmp

Изменение файла lab7-1.asm, после которого программа выводит сначала сообщение 2, потом сообщение 1 (рис. [3](#fig:003)).

%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

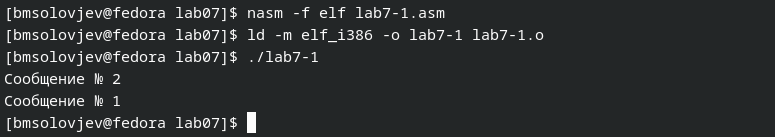


Figure 3: Другая программа с функцией jmp

Теперь изменяю файл lab7-1.asm таким образом, чтобы она выводил все сообщения (рис. [4](#fig:004)).

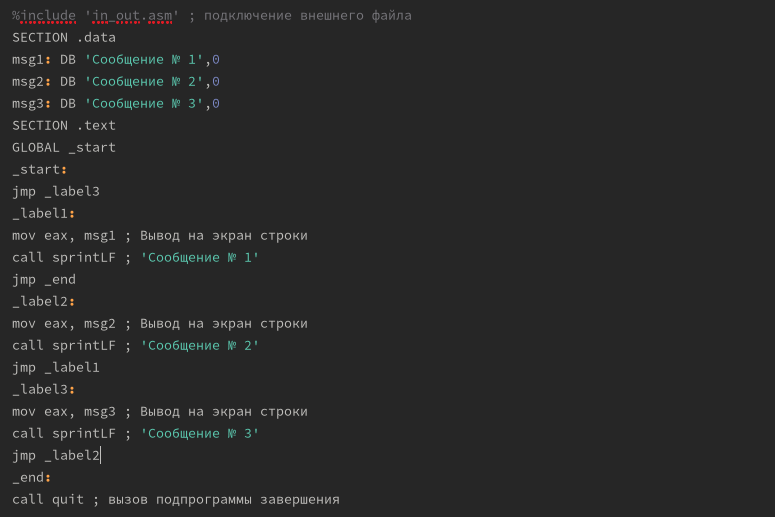


Figure 4: Код, выводящий все сообщения

Проверка работы моего кода (рис. [5](#fig:005)).

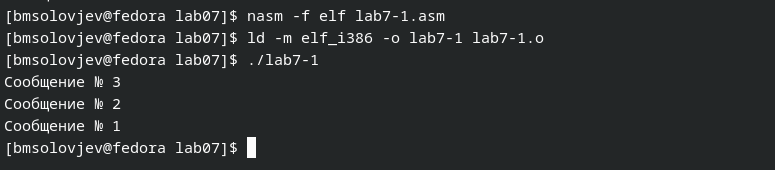


Figure 5: Проверка работы моего кода

Код программы, сравнивающей 3 числа (рис. [6](#fig:006)).

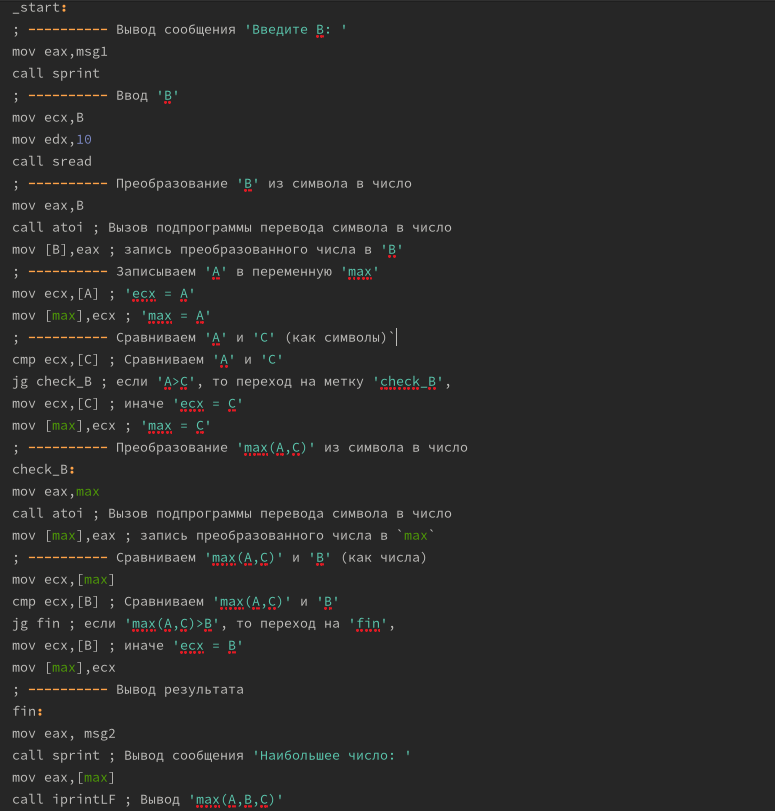


Figure 6: Код, сравнивающий 3 числа

Проверка работы кода, сравнивающего 3 числа (рис. [7](#fig:007)).

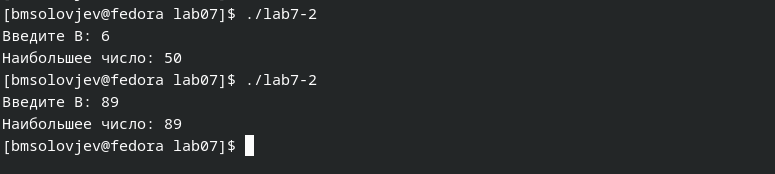


Figure 7: Проверка кода

Создаю файл листинга и открываю его с помощью текстового редактора. 14 строка вычитает из значения, находящегося в регистре eax, значение, находящееся в регистре ebx. 15 строка берёт из стека значение, адрес которого находится в регистре ebx. 16 строка выполняет возврат из процедуры (рис. [7](#fig:007)).

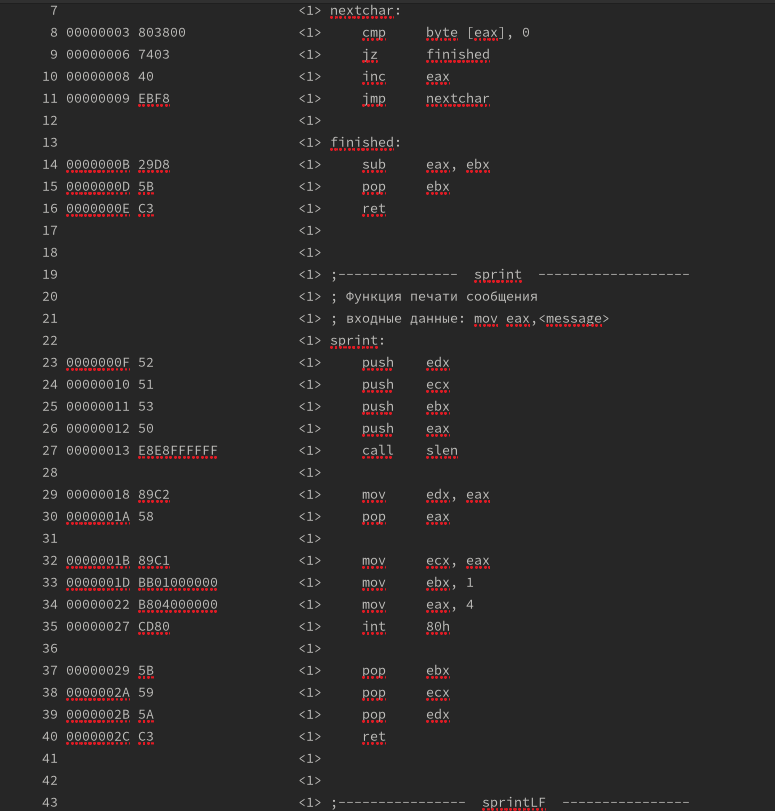


Figure 8: Файл листинга

Удаляю один операнд из инструкции mov и получаю ошибку (рис. [9](#fig:009)).

%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] УДАЛИЛ ТУТ  
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

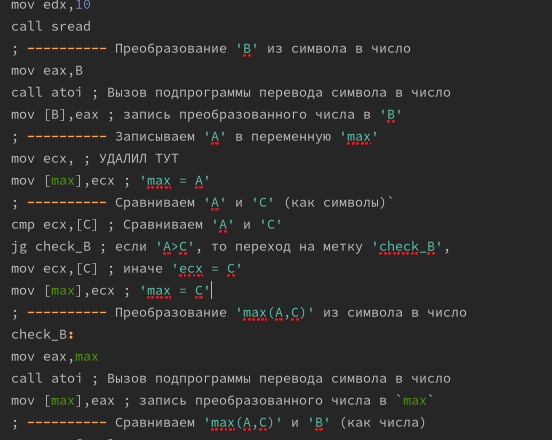


Figure 9: Удаление операнда

# 4 Самостоятельная работа

## 4.1 1 задание

Мой код для программы, находящей наименьшую из 3 переменных 94б 5б 58 (рис. [10](#fig:010)).

%include 'in\_out.asm'  
section .data  
msg2 db "Наименьшее из трёх чисел:",0h  
A dd '94'  
B dd '5'  
C dd '58'  
section .bss  
min resb 10  
section .text  
global \_start  
\_start:  
mov eax,C  
call atoi ; Вызов подпрограммы перевода символа в число  
mov [C],eax ; запись преобразованного числа в 'C'  
; ---------- Записываем 'A' в переменную 'min'  
mov ecx,[A] ; 'ecx = A'  
mov [min],ecx ; 'max = A'  
; ---------- Сравниваем 'A' и 'B' (как символы)  
cmp ecx,[B] ; Сравниваем 'A' и 'B'  
jl check\_C ; если 'A<B', то переход на метку 'check\_C',  
mov ecx,[B] ; иначе 'ecx = B'  
mov [min],ecx ; 'min = B'  
; ---------- Преобразование 'max(A,C)' из символа в число  
check\_C:  
mov eax,min  
call atoi ; Вызов подпрограммы перевода символа в число  
mov [min],eax ; запись преобразованного числа в min  
; ---------- Сравниваем 'min(A,B)' и 'C' (как числа)  
mov ecx,[min]  
cmp ecx,[C] ; Сравниваем 'min(A,B)' и 'C'  
jl fin ; если 'min(A,B)<C', то переход на 'fin',  
mov ecx,[C] ; иначе 'ecx = C'  
mov [min],ecx  
; ---------- Вывод результата  
fin:  
mov eax, msg2  
call sprint ; Вывод сообщения 'Наименьшее число: '  
mov eax,[min]  
call iprintLF ; Вывод 'min(A,B,C)'  
call quit ; Выход

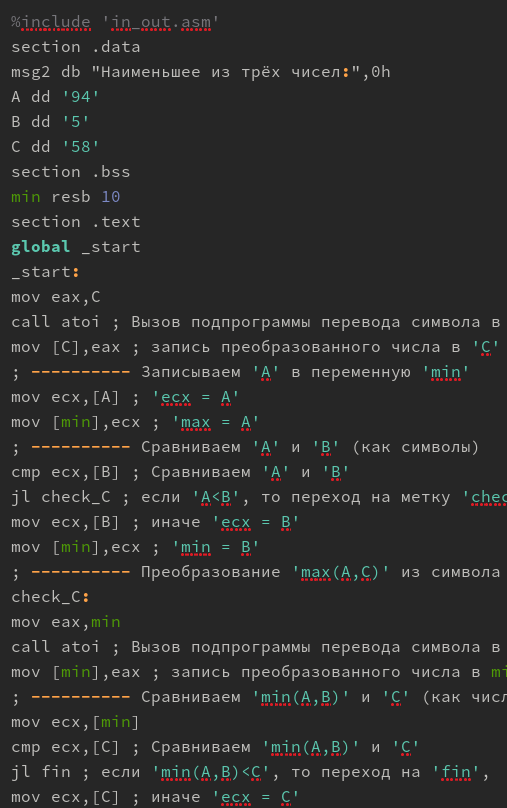


Figure 10: Код программы, находящей min

Результат работы моего кода (рис. [11](#fig:011)).

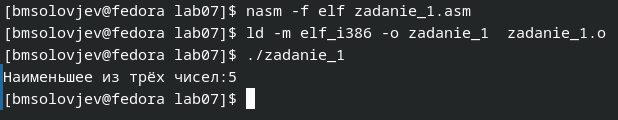


Figure 11: Проверка кода

## 4.2 2 задание

Мой вариант - 3, поэтому я написал код, который запрашивает 2 числа, и если первое не равно 3, то выводит второе число + 1, а если равно 3, то умножает первое число на 3(рис. [12](#fig:012)).

%include 'in\_out.asm'  
section .data  
msg1 db 'Введите х: ',0h  
msg2 db 'Введите а: ',0h  
section .bss  
x resb 10  
a resb 10  
n resb 10  
section .text  
global \_start  
\_start:  
; ---------- Вывод сообщения 'Введите x: '  
mov eax,msg1  
call sprint  
; ---------- Ввод 'x'  
mov ecx,x  
mov edx, 80  
call sread  
; ---------- Преобразование 'x' из символа в число  
mov eax,x  
call atoi ; Вызов подпрограммы перевода символа в число  
mov [x],eax ; запись преобразованного числа в 'x'  
  
; ---------- Вывод сообщения 'Введите a: '  
mov eax,msg2  
call sprint  
; ---------- Ввод 'a'  
mov ecx,a  
call sread  
; ---------- Преобразование 'a' из символа в число  
mov eax,a  
call atoi ; Вызов подпрограммы перевода символа в число  
mov [a],eax ; запись преобразованного числа в 'a'  
  
; --------- Функция  
mov eax, 3  
cmp eax, [x]  
jne fin  
jmp fin1  
  
; ---------- Вывод результата  
fin1:  
mov eax, [a]  
add eax, 1   
mov [a],eax  
mov eax, [a] ;  
call iprintLF ;   
  
call quit ; Выход  
  
fin:  
mov eax, [x]  
mov ebx, 3  
mul ebx  
mov [x],eax  
mov eax, [x] ;  
call iprintLF ;   
  
call quit ; Выход

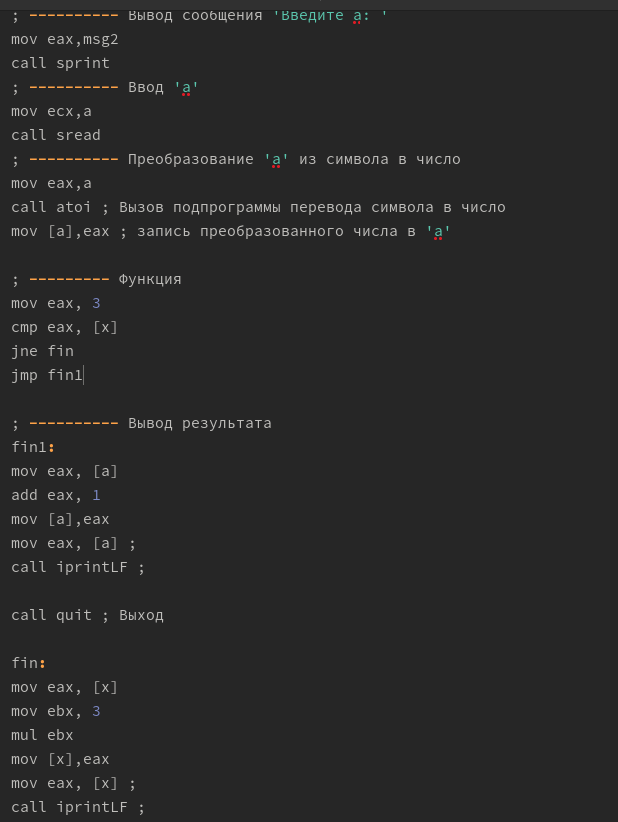


Figure 12: Код моей программы

Проверка работы кода (рис. [13](#fig:013)).

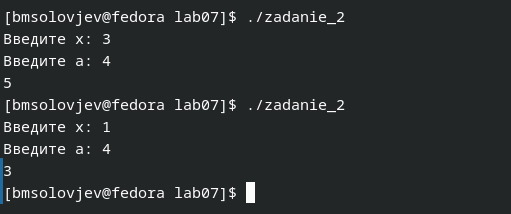


Figure 13: Проверка кода

# 5 Выводы

Выполнив данную лабораторную работу, я изучил команды условного и безусловного перехода. Приобрёл навыки написания программ с использованием переходов. Познакомился с назначением и структурой файла листинга.

# Список литературы