РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 7

дисциплина: Архитектура компьютера

Студент: Ибрагимов Гаджимурад Шамильевич

Группа: НКАбд-02-25

МОСКВА

2025 г.

СОДЕРЖАНИЕ

1. Цель работы

Данная лабораторная работа направлена на освоение механизмов условных и безусловных переходов, а также работы с листингом программы в рамках ассемблера NASM.

2. Теоретическое введение

2.1 Безусловные переходы

Безусловный переход — это переход, который выполняется всегда. Безусловный переход осуществляется с помощью команды JMP. У этой команды один операнд, который может быть непосредственным адресом (меткой), регистром или ячейкой памяти, содержащей адрес. Существуют также «дальние» переходы — между сегментами, однако здесь мы их рассматривать не будем. Примеры безусловных переходов:

```
jmp metka     ;Переход на метку
jmp bx        ;Переход по адресу в BX
jmp word[bx] ;Переход по адресу, содержащемуся в памяти по адресу в BX
```

2.2 Условные переходы

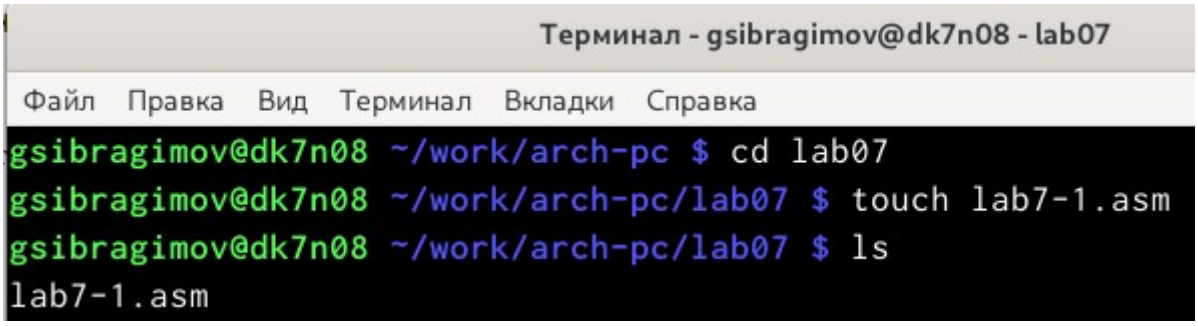
Условный переход осуществляется, если выполняется определённое условие, заданное флагами процессора (кроме одной команды, которая проверяет CX на равенство нулю). Как вы помните, состояние флагов изменяется после выполнения арифметических, логических и некоторых других команд. Если условие не выполняется, то управление переходит к следующей команде.

Команда	Переход, если	Условие перехода
JZ/JE	нуль или равно	ZF=1
JNZ/JNE	не нуль или не равно	ZF=0
JC/JNAE/JB	есть переполнение/не выше и не равно/ниже	CF=1
JNC/JAE/JNB	нет переполнения/выше или равно/не ниже	CF=0
JP	число единичных бит чётное	PF=1
JNP	число единичных бит нечётное	PF=0
JS	знак равен 1	SF=1
JNS	знак равен 0	SF=0
JO	есть переполнение	OF=1
JNO	нет переполнения	OF=0
JA/JNBE	выше/не ниже и не равно	CF=0 и ZF=0
JNA/JBE	не выше/ниже или равно	CF=1 или ZF=1
JG/JNLE	больше/не меньше и не равно	ZF=0 и SF=OF
JGE/JNL	больше или равно/не меньше	SF=OF
JL/JNGE	меньше/не больше и не равно	SF≠OF
JLE/JNG	меньше или равно/не больше	ZF=1 или SF≠OF
JCXZ	содержимое CX равно нулю	CX=0

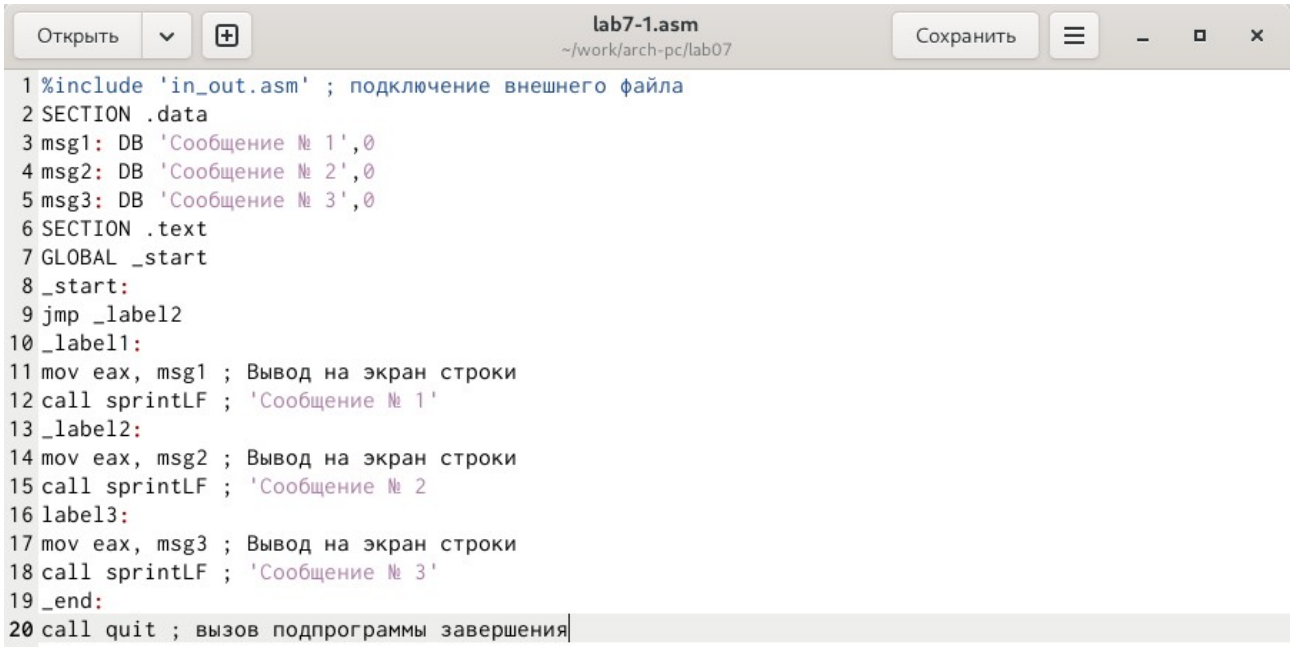
3. Описание выполнения лабораторной работы

Реализация переходов в NASM

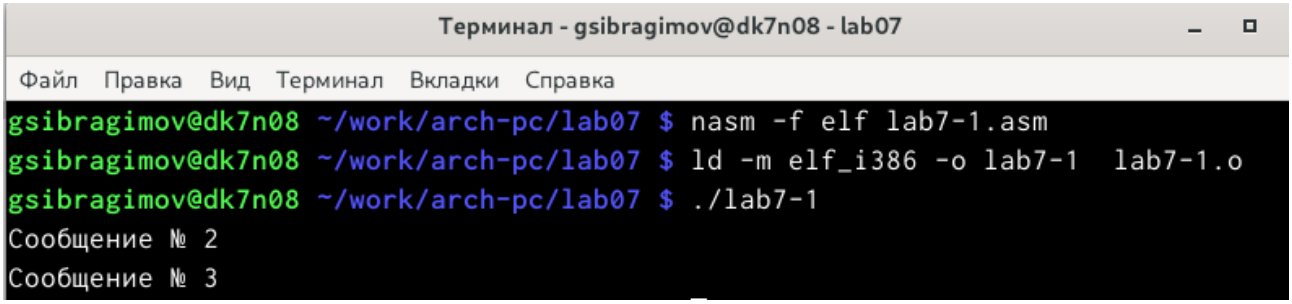
1. Создаем каталог для программам лабораторной работы № 7, перейдите в него и создайте файл lab7-1.asm:



2. 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 лекции



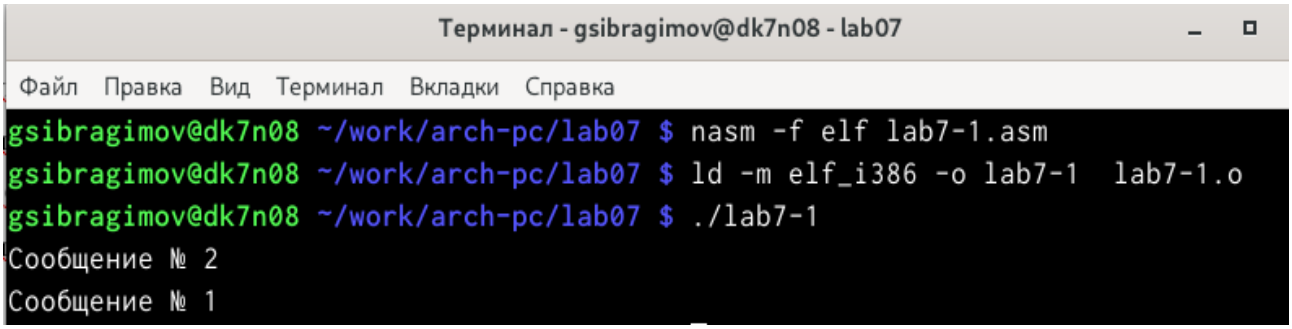
Создаем исполняемый файл и запускаем его. Результат работы данной программы будет следующим:



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

3. 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.
Изменим текст программы в соответствии с листингом данным в лекции.

Создаем исполняемый файл и проверяем его работу:



3.1 Выполним следующую задачу:

Создайте исполняемый файл и проверьте его работу.
Измените текст программы добавив или изменив инструкции jmp, чтобы вывод программы был следующим:

```
user@dk4n31:~$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
user@dk4n31:~$
```

Переписываем текст файла lab7-1.asm:

lab7-1.asm — KWrite

Файл Правка Выделение Вид Переход Сервис Настройка Справка

Создать Открыть... Сохранить Сохранить как... Отменить действие Повторить

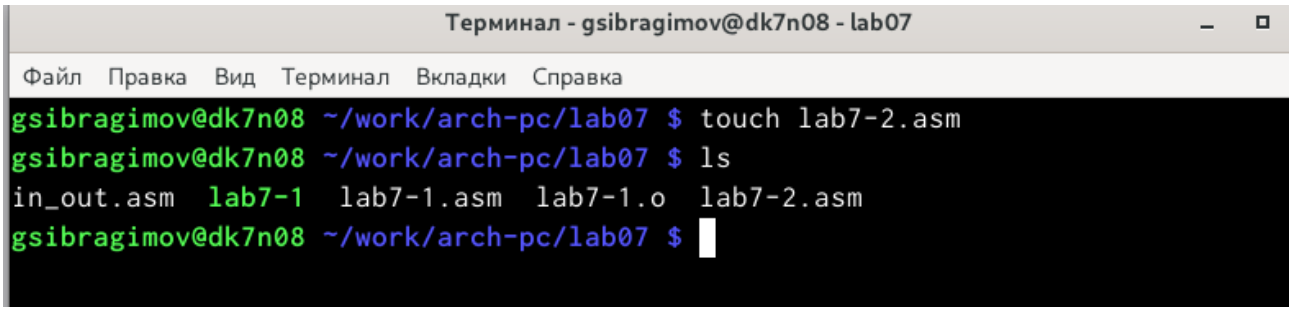
```
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label3
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintLF   ; 'Сообщение № 3'
jmp _label2
_end:
call quit ; вызов подпрограммы завершения
```

Создаем исполняемый файл и проверяем его работу:

```
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1  lab7-1.o
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $
```

4. 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 клавиатуры.

Создаем файл lab7-2.asm в каталоге lab07.



Файл с текстом программы:

Открыть

lab7-2.asm
~/work/arch-pc/lab07

```
%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:
mov eax,msg1
call sprint
mov ecx,B
mov edx,10
call sread
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'B'
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'
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
```

Запускаем программу:

```
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2  lab7-2.o
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 69
Наибольшее число: 69
```

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

Создаем файл листинга для программы из файла lab7-2.asm

Терминал - gsibragimov@dk7n08 - lab07

Файл Правка Вид Терминал Вкладки Справка

```
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ ls
in_out.asm  lab7-1.asm  lab7-2      lab7-2.lst
lab7-1      lab7-1.o    lab7-2.asm  lab7-2.o
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $
```

Текст листинга:

Терминал - gsibragimov@dk7n08 - lab07

Файл Правка Вид Терминал Вкладки Справка

```
lab7-2.lst        [----]  0 L:[185+ 0 185/221] *(11284/14060b) 0032 0x020                [*][X]
   10                                 section .text
   11                                 global _start
   12                                 _start:
   13 000000E8 B8[00000000]           mov eax,msg1
   14 000000ED E81DFFFFFF             call sprint
   15 000000F2 B9[0A000000]           mov ecx,B
   16 000000F7 BA0A000000             mov edx,10
   17 000000FC E842FFFFFF             call sread
   18 00000101 B8[0A000000]           mov eax,B
   19 00000106 E891FFFFFF             call atoi ; Вызов подпрограммы перевода символа в число
   20 0000010B A3[0A000000]           mov [B],eax ; запись преобразованного числа в 'B'
   21 00000110 8B0D[35000000]         mov ecx,[A] ; 'ecx = A'
   22 00000116 890D[00000000]         mov [max],ecx ; 'max = A'
   23                                 ; ---------- Сравниваем 'A' и 'C' (как символы)
   24 0000011C 3B0D[39000000]         cmp ecx,[C] ; Сравниваем 'A' и 'C'
   25 00000122 7F0C                   jg check_B ; если 'A>C', то переход на метку 'check_B',
   26 00000124 8B0D[39000000]         mov ecx,[C] ; иначе 'ecx = C'
   27 0000012A 890D[00000000]         mov [max],ecx ; 'max = C'
   28                                 ; ---------- Преобразование 'max(A,C)' из символа в число
   29                                 check_B:
   30 00000130 B8[00000000]           mov eax,max
   31 00000135 E862FFFFFF             call atoi ; Вызов подпрограммы перевода символа в число
```

Строка 15

B9 – однобайтовый опкод

по сути инструкция mov ECX, imm32(32 битное число)

Строка 16

BA – Опкод

по сути инстуркция mov EDX,imm32(32 битное число)

Строка 17

E8 – опкод вызова функции

6. Проведем тест. Изменим програмный код. Удалим операнд C

```
23  ; ---------- Сравниваем 'A' и 'C' (как символы)
24  cmp ecx, ; Сравниваем 'A' и 'C'
25  jg check_B ; если 'A>C', то переход на метку 'check_B',
26  mov ecx,[C] ; иначе 'ecx = C'
27  mov [max],ecx ; 'max = C'
```

Создаем файл листинга и просматриваем:

```
24                              cmp ecx, ; Сравниваем 'A' и 'C'
24          ******************       error: invalid combination of opcode and operands
```

На месте, где удален операнд выводится ошибка. Это удобно для мониторинга ошибок и неточностей.

4. Описание выполнения самостоятельной работы

1. Напишите программу нахождения наименьшей из 3 целочисленных переменных *a*,*b* и c

Мои значения переменных: 13 84,32,77

Создаем файл 1.asm и прописываем следующие команды:

Открыть ▾ 1.asm ~/work/arch-pc/lab07 Сохранить

```
%include 'in_out.asm'
section .data
msg1 db 'Программа для вычисления наименьшего числа: ',0h
A dd 84
B dd 32
C dd 77
section .bss
min resb 10

section .text
global _start
_start:

 mov eax, msg1
 call sprint

mov ecx,[A] ; 'ecx = A'
mov [min], ecx ; 'min = A'

cmp ecx,[B]
jl check_C
mov ecx,[B] ; 'ecx = B'
mov [min],ecx ; 'min = B

check_C:
mov ecx,[min]
cmp ecx, [C]
jl final
mov ecx, [C]
mov [min], ecx
final:
mov eax, [min]
call iprintLF
call quit
```

Запускаем программу:

```
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ nasm -f elf 1.asm
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o 1  1.o
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ ./1
Программа для вычисления наименьшего числа: 32
```

2. Напишите программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции $f(x)$ и выводит результат вычислений. Вид функции $f(x)$ выбрать из таблицы

Мой вид функции и значения:

13	$\begin{cases} a-7, & a \geq 7 \\ ax, & a < 7 \end{cases}$	(3;9)	(6;4)

Открыть

2.asm
~/work/arch-pc/lab07

```
%include 'in_out.asm'
section .data
msg1 db 'Введите x: ',0h
msg2 db 'Введите a: ',0h
section .bss
x resb 10
a resb 10

section .text
global _start
_start:
mov eax,msg1
call sprint
mov ecx,x
mov edx,10
call sread
mov eax,x
call atoi ; Вызов подпрограммы перевода символа в число
mov [x],eax ;

mov eax,msg2
call sprint
mov ecx,a
mov edx,10
call sread
mov eax,a
call atoi ; Вызов подпрограммы перевода символа в число
mov [a],eax ;

mov ecx, [a]
cmp ecx, 7
jae var1
mov eax, [a]
mov ebx, [x]
mul ebx
call iprintLF
 call quit

var1:
mov eax, [a]
sub eax, 7
call iprintLF
 call quit
```

Запускаем программу:

```
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ nasm -f elf  2.asm
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o 2  2.o
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ ./2
Введите x: 7
Введите a: 7
0
```

Для больше или равно 7 отрабатывает корректно , то есть a - 7

```
gsibragimov@dk7n08 ~/work/arch-pc/lab07 $ ./2
Введите x: 55
Введите a: 3
165
```

Для меньше 7 выполнилась программа корректно также , то есть ax

5. ВЫВОД

Я ознакомился с безусловными и условными переходами, а также научился создавать файлы листинга для просмотра и отладки программ.

6. ИСТОЧНИКИ

1. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. (Классика Computer Science).

2. FASMWORLD
https://fasmworld.ru/uchebnyj-kurs/016-uslovnye-i-bezuslovnye-perexody/