

# **Лабораторная работа № 7**

**Архитектура компьютеров**

Тойчубекова Асель Нурлановна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>8</b>
3.1	Команды безусловного перехода . . . . .	8
3.2	Команды условного перехода . . . . .	8
3.3	Инструкция str . . . . .	9
3.4	Описание команд условного перехода . . . . .	10
3.5	Файл листинга и его структура . . . . .	10
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>12</b>
4.1	Задание для самостоятельной работы . . . . .	19
<b>5</b>	<b>Выводы</b>	<b>28</b>
	<b>Список литературы</b>	<b>29</b>

## Список иллюстраций

3.1	РИС.1 Регистр флагов . . . . .	9
3.2	РИС.2 Команды условного перехода . . . . .	10
4.1	РИС.3 Создание каталога и файла. . . . .	12
4.2	РИС.4 Редактирование файла . . . . .	12
4.3	РИС.5 Запуск исполняемого файла . . . . .	13
4.4	РИС.7 Запуск исполняемого файла . . . . .	14
4.5	РИС.8 Редактирование файла . . . . .	14
4.6	РИС.9 Запуск исполняемого файла . . . . .	14
4.7	РИС.10 Редактирование файла . . . . .	15
4.8	РИС.11 Запуск исполняемого файла . . . . .	16
4.9	РИС.11 Запуск исполняемого файла . . . . .	16
4.10	РИС.12 Создание файла листинга . . . . .	16
4.11	РИС.13 Изучение файла листинга . . . . .	17
4.12	РИС.14 Файл листинга . . . . .	18
4.13	РИС.15 Ошибка трансляции lab7-2 . . . . .	19
4.14	РИС.15_1 Файл листинга с ошибкой . . . . .	19
4.15	РИС.16 Создание файла . . . . .	21
4.16	РИС.17 Редактирование файла . . . . .	22
4.17	РИС.18 Запуск исполняемого файла . . . . .	23
4.18	РИС.19 Создание файла . . . . .	25
4.19	РИС.20 Редактирование файла . . . . .	26
4.20	РИС.21 Запуск исполняемого файла . . . . .	27
4.21	РИС.22 Запуск исполняемого файла . . . . .	27

## **Список таблиц**

# 1 Цель работы

Целью лабораторной работы №7 является изучение команд условного и безусловного переходов. А также приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

## 2 Задание

- Понять работу команд управления или команды перехода: условный переход, безусловный переход;
  - Изучить работу регистра флагов: Carry Flag - Флаг переноса, Parity Flag - Флаг чётности, Auxiliary Carry Flag -Вспомогательный флаг переноса, Zero Flag - Флаг нуля, Sign Flag - Флаг знака, Overflow Flag - Флаг переполнения;
  - Изучить работу инструкции `cmp`;
  - Рассмотреть файл листинга и его структуру;
  - Написать программу с использованием инструкции `jmp` опираясь на пример;
  - Написать программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: A, B, C опираясь на пример;  
-Задание для самостоятельной работы:
1. Напишите программу нахождения наименьшей из 3 целочисленных переменных a, b и c. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу.
  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 7.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу для значений  $x$  и  $a$  из 7.6.

## 3 Теоретическое введение

Для реализации ветвлений в ассемблере используются так называемые команды передачи управления или команды перехода. Можно выделить 2 типа переходов: -условный переход-выполняется или не выполняется переход в соответствии с условием; -безусловный переход-выполняется переход в указанную точку программы.

### 3.1 Команды безусловного перехода

Безусловный переход выполняется инструкцией `jmp`, которая включает в себя адрес перехода, куда следует передать управление:

`jmp < адрес_перехода >`

### 3.2 Команды условного перехода

В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов. Регистр флагов – это очень важный регистр процессора, который используется при выполнении большинства команд. Регистр флагов носит название EFLAGS. Это 32-разрядный регистр. Однако старшие 16 разрядов используются при работе в защищённом режиме, и пока мы их рассматривать не будем. К младшим 16 разрядам этого регистра можно обращаться как к отдельному регистру с именем FLAGS. Именно этот регистр мы и рассмотрим в этом разделе. Каждый бит в регистре FLAGS является флагом. Флаг – это один



или несколько битов памяти, которые могут принимать двоичные значения (или комбинации значений) и характеризуют состояние какого-либо объекта. Обычно флаг может принимать одно из двух логических значений. Поскольку в нашем случае речь идёт о бите, то каждый флаг в регистре может принимать либо значение 0, либо значение 1. Флаги устанавливаются в 1 при определённых условиях, или установка флага в 1 изменяет поведение процессора. На РИС.1 показано, какие флаги находятся в разрядах регистра FLAGS.

Регистр флагов		
Бит №	Имя Флага	Назначение
0	CF	Флаг переноса. Был ли перенос из или заем в старший разряд.
1	PF	Флаг четности. 1, если результат операции содержит четное количество единиц.
4	AF	Флаг вспомогательного переноса
6	ZF	Флаг нуля. 1, если результат операции равен 0.
7	SF	Флаг знака. Показывает знак результата операции (1- отрицательный, 0-положительный)
8	TF	Флаг трассировки. Обеспечивает возможность работы процессора в пошаговом режиме.
9	IF	Флаг внешних прерываний. Если IF=1, прерывание разрешается, IF=0 – блокируется.
10	DF	Флаг направления. Используется командами обработки строк. DF=1 – прямое направление (от меньших адресов к большим). DF=0 – обратное направление
12	OF	Флаг переполнения.

Рис. 3.1: РИС.1 Регистр флагов

### 3.3 Инструкция `cmp`

Инструкция `cmp` позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения:

`cmp < операнд_1 >, < операнд_2 >`

Эта инструкция никуда не записывает результат и единственным результатом команды сравнения является формирование флагов.

### 3.4 Описание команд условного перехода

Команда условного перехода имеет вид:

j< мнемоника перехода > label

Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов. На РИС.2 представлены команды условного перехода, которые обычно ставятся после команды сравнения стр.

Код команды	Реальное условие	Условие для CMP	Код команды	Реальное условие	Условие для CMP
JA	CF=0 и ZF=0	если выше	JG	ZF=0 и SF=OF	если больше
JAЕ JNC	CF=0	если выше или равно если нет переноса	JGE	SF=OF	если больше или равно
JB JC	CF=1	если ниже если перенос	JL	SF<>OF	если меньше
JBE	CF=1   ZF=1	если ниже или равно	JLE	ZF=1   SF<>OF	если меньше или равно
JE JZ	ZF=1	если равно если ноль	JNE JNZ	ZF=0	если не равно если не ноль
JO	OF=1	если есть переполнение	JNO	OF=0	если нет переполнения
JS	SF=1	если есть знак	JNS	SF=0	если нет знака
JP	PF=1	если есть четность	JNP	PF=0	если нет четности

Рис. 3.2: РИС.2 Команды условного перехода

### 3.5 Файл листинга и его структура

Листинг — 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 имприсваивается).

## 4 Выполнение лабораторной работы

Создадим каталог для программ лабораторной работы №7, затем перейдем в него и создадим в нем файл lab7-1.asm, с которым будем работать. Введя команду ls удостоверимся что файл был создан.(РИС.3)

```
antoychubekova@dk6n55 ~ $ mkdir ~/work/arch-pc/lab07
antoychubekova@dk6n55 ~ $ cd ~/work/arch-pc/lab07
antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ touch lab7-1.asm
antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ ls
lab7-1.asm
```

Рис. 4.1: РИС.3 Создание каталога и файла.

Далее введем текст программы, в котором используется инструкция jmp в файл lab7-1.asm (РИС.4)

```
lab7-1.asm [-M--] 41 L:[ 1+19 20/ 20] *(643 / 643b) <EOF>
%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"
_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:
call quit ; вызов подпрограммы завершения
```

Рис. 4.2: РИС.4 Редактирование файла

Создадим исполняемый файл и запустим его. Видим что исполняемая инструкция `jmp _label2` поменяла порядок исполнения инструкций и позволила(перепрыгнула) начать выполнение инструкции с меткой `_label2`, пропустив вывод первого сообщения.(РИС.5)

```
antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 3
```

Рис. 4.3: РИС.5 Запуск исполняемого файла

Изменим текст программы так, чтобы на выводила сначала ‘Сообщение No 2’, потом ‘Сообщение № 1’ и завершала работу.(РИС.6)

```
lab7-1.asm      [-M--]  0 L:[ 1+22 23/ 23] *(665 / 665b) <EOF>
%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 ; "Сообщение No 2"
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; "Сообщение No 3"
_end:
call quit ; вызов подпрограммы завершения
```

Создадим исполняемый файл и запустим его. Мы видим, что все правильно работает.(РИС.7)

```

antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 1

```

Рис. 4.4: РИС.7 Запуск исполняемого файла

Изменим текст программы так, чтобы сообщения выводились в попятке убывания, Сообщение№3->Сообщение№2->Сообщение№3.(РИС.8)

```

lab7-1.asm [----] 11 L:[ 1+20 21/ 24] *(601 / 677b) 0010
#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 ; 'Сообщение No 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение No 3'
jmp _label2
_end:
call quit ; вызов подпрограммы завершения

```

Рис. 4.5: РИС.8 Редактирование файла

Создадим исполняемый файл и запустим его. Видим, что все правильно выводится.(РИС.9)

```

antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 3
Сообщение No 2
Сообщение No 1

```

Рис. 4.6: РИС.9 Запуск исполняемого файла

Созадем файл lab7-2.asm в исходном каталоге, далее напишем в него текст программы, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: A, B, C. (РИС.10)

```
lab7-2.asm [----] 18 L: [ 1+38 39/ 49] *(1379/1744b) 1085 0x43D
%include "input.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 '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
```

Рис. 4.7: РИС.10 Редактирование файла

Создадим исполняемый файл и запустим его. Проверяем его работу на числах 1 и 60 и видим, что все правильно работает. (РИС.11) и (РИС.11\_1)

```

antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ ./lab7-2
Введите В: 1
Наибольшее число: 50

```

Рис. 4.8: РИС.11 Запуск исполняемого файла

```

antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ ./lab7-2
Введите В: 60
Наибольшее число: 60

```

Рис. 4.9: РИС.11 Запуск исполняемого файла

Создайте файл листинга для программы из файла lab7-2.asm, введя команду:  
 nasm -f elf -l lab7-2.lst lab7-2.asm.(РИС.12)

```

antoychubekova@dk6n55 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm

```

Рис. 4.10: РИС.12 Создание файла листинга

Открое файл листинга lab7-2.lst с помощью текстового редактора mcedit и  
 внимательно изучим его формат и содержание.(РИС.13)



```

lab7-2.lst      [---]  0 L:[169+27 196/225] *(11987/14459b) 0032 0x020  [*][X]
168 000000E0 B801000000 <I> mov     eax, 1
169 000000E5 CD80 <I> int     80h
170 000000E7 C3 <I> ret
      2
      3 section .data
      4 msg1 db 'Введите B: ',0h
      5 00000000 D092D0B2D0B5D0B4D0-
      6 00000009 B8D182D0B520423A20-
      7 00000012 00
      8 00000013 D09DD0B0D0B8D0B1D0-
      9 0000001C BED0BB018CD188D0B5-
     10 00000025 D0B520D187D0B8D181-
     11 0000002E D0BB00BE3A2000
     12
     13 A dd '20'
     14 C dd '50'
     15 section .bss
     16 max resb 10
     17 B resb 10
     18 section .text
     19 global _start
     20 _start:
     21 ; ----- Вывод сообщения 'Введите B: '
     22 000000E8 B8[00000000] mov     eax,msg1
     23 000000ED E81DFFFFFF call    sprint
     24 ; ----- Ввод 'B'
     25 000000F2 B9[0A000000] mov     ecx,B
     26 000000F7 BA0A000000 mov     edx,10
     27 000000FC E842FFFFFF call    sread
     28 ; ----- Преобразование 'B' из символа в число
     29 00000101 B8[0A000000] mov     eax,B
     30 00000106 E891FFFFFF call    atoi ; Вызов подпрограммы перевода символа в чи
     31 0000010B A3[0A000000] mov     [B],eax ; запись преобразованного числа в 'B'
     32 ; ----- Записываем 'A' в переменную 'max'
     33 00000110 8B0D[35000000] mov     ecx,[A] ; 'ecx = A'
     34 00000116 890D[00000000] mov     [max],ecx ; 'max = A'
     35 ; ----- Сравниваем 'A' и 'C' (как символы)
     36 0000011C 3B0D[39000000] cmp     ecx,[C] ; Сравниваем 'A' и 'C'
     37 00000122 7F0C jg     check_B ; если 'A>C', то переход на метку 'check_
     38 00000124 8B0D[39000000] mov     ecx,[C] ; иначе 'ecx = C'
     39 0000012A 890D[00000000] mov     [max],ecx ; 'max = C'
     40 ; ----- Преобразование 'max(A,C)' из символа в
     41 check_B:
     42 00000130 B8[00000000] mov     eax,max

```

Рис. 4.11: РИС.13 Изучение файла листинга

Подробно объясним содержание представленных трех строчках на РИС.14

```

lab7-2.lst      [---]  0 L:[169+27 196/225] *(11987/14459b) 0032 0x020  [*][X]
168 000000E0 B801000000 <1> mov     eax, 1
169 000000E5 CD80 <1> int     80h
170 000000E7 C3 <1> ret
      2
      3 00000000 D092D0B2D0B5D0B4D0- msg1 db 'Введите B: ',0h
      3 00000009 B8D182D0B520423A20-
      3 00000012 00
      4 00000013 D09DD0B0D0B8D0B1D0- msg2 db "Наибольшее число: ",0h
      4 0000001C BED0BB018CD188D0B5-
      4 00000025 D0B520D187D0B8D181-
      4 0000002E D0BBD0BE3A2000
      5 00000035 32300000 A dd '20'
      6 00000039 35300000 C dd '50'
      7
      8 00000000 <res Ah> section .bss
      9 0000000A <res Ah> max resb 10
      10 B resb 10
      11 section .text
      12 global _start
      13 _start:
      14 ; ----- Вывод сообщения 'Введите B: '
      14 000000E8 B8[00000000] mov     eax,msg1
      15 000000ED E81DFFFFFF call    sprint
      16 ; ----- Ввод 'B'
      17 000000F2 B9[0A000000] mov     ecx,B
      18 000000F7 BA0A000000 mov     edx,10
      19 000000FC E842FFFFFF call    sread
      20 ; ----- Преобразование 'B' из символа в число
      21 00000101 B8[0A000000] mov     eax,B
      22 00000106 E891FFFFFF call    atoi ; Вызов подпрограммы перевода символа в чи
      23 0000010B A3[0A000000] mov     [B],eax ; запись преобразованного числа в 'B'
      24 ; ----- Записываем 'A' в переменную 'max'
      25 00000110 8B0D[35000000] mov     ecx,[A] ; 'ecx = A'
      26 00000116 890D[00000000] mov     [max],ecx ; 'max = A'
      27 ; ----- Сравниваем 'A' и 'C' (как символы)
      28 0000011C 3B0D[39000000] cmp     ecx,[C] ; Сравниваем 'A' и 'C'
      29 00000122 7F0C jg     check_B ; если 'A>C', то переход на метку 'check_
      30 00000124 8B0D[39000000] mov     ecx,[C] ; иначе 'ecx = C'
      31 0000012A 890D[00000000] mov     [max],ecx ; 'max = C'
      32 ; ----- Преобразование 'max(A,C)' из символа в
      33 check_B:
      34 00000130 B8[00000000] mov     eax,max

```

Рис. 4.12: РИС.14 Файл листинга

строка 196

- 21 - номер строки в подпрограмме
- 00000101 - адрес
- B8 [0A000000] - машинный код
- mov ecx,B - код программы - копирует B в ecx.

строка 197

- 22 - номер строки в подпрограмме
- 00000106 - адрес
- E891FFFFFF - машинный код.

строка 198

- 23 - номер строки в подпрограмме
- 0000010B - адрес

- A3[0A000000] - машинный код.

Открыл файл с программой lab7-2.asm и в инструкции с двумя операндами удалив один операнд. Выполнил трансляцию с получением файла листинга.(РИС.15) и (РИС.15\_1)

```
antoychubekova@dk5n59 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:23: error: expression syntax error
antoychubekova@dk5n59 ~/work/arch-pc/lab07 $
```

Рис. 4.13: РИС.15 Ошибка трансляции lab7-2

```
16 ; ----- Ввод 'B'
17 000000F2 B9[0A000000] mov ecx,B
18 000000F7 BA0A000000 mov edx,10
19 000000FC E842FFFFFF call spread
20 ; ----- Преобразование 'B' из символа в число
21 00000101 B8[0A000000] mov eax,B
22 00000106 E891FFFFFF call atoi ; Вызов подпрограммы перевода символа в число
23 0000010B A3[0A000000] mov [B],eax ; запись преобразованного числа в 'B'
24 ; ----- Записываем 'A' в переменную 'max'
25 mov ecx, ; 'ecx = A'
25 *****
26 00000110 890D[00000000] error: invalid combination of opcode and operands
26 mov [max],ecx ; 'max = A'
27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 00000116 3B0D[39000000] cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 0000011C 7F0C jg check_B ; если 'A>C', то переход на метку 'check_B',
30 0000011E 8B0D[39000000] mov ecx,[C] ; иначе 'ecx = C'
31 00000124 890D[00000000] mov [max],ecx ; 'max = C'
32 ; ----- Преобразование 'max(A,C)' из символа в число
33 check_B:
34 0000012A B8[00000000] mov eax,max
35 0000012F E868FFFFFF call atoi ; Вызов подпрограммы перевода символа в число
36 00000134 A3[00000000] mov [max],eax ; запись преобразованного числа в 'max'
37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
38 00000139 8B0D[00000000] mov ecx,[max]
39 0000013F 3B0D[0A000000] cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
```

Рис. 4.14: РИС.15\_1 Файл листинга с ошибкой

Объектный файл не смог создаться из-за ошибки. Но получился листинг, где выделено место ошибки.

## 4.1 Задание для самостоятельной работы

Создадим файл lab7-3.asm в котором будем работать.(РИС.16) Напишем программу нахождения наименьшей из 3 целочисленных переменных a,b и c. Значения переменных-81,22,72.(РИС.17) Сама программа выглядит следующим образом:

```

%include 'in_out.asm'

section .data
msg1 db "Наибольшее число:",0h
A dd '81'
B dd '22'
C dd '72'

```

```

section .bss
min resb 10

```

```

section .text
global _start
_start:

```

```

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

```

```

    mov ecx,[A] mov [min],ecx
    cmp ecx,[C]
    jl check_B
    mov ecx,[C] mov [min],ecx

```

```

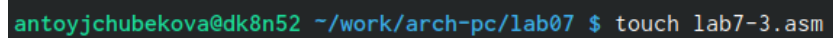
    check_B:
    mov eax,min
    call atoi
    mov [min],eax

```

```
    mov ecx,[min]
cmp ecx,[B] jl fin  mov ecx,[B] mov [min],ecx
```

```
fin:
mov eax, msg1
call sprint
mov eax,[min]
```

```
call iprintLF  call quit
```



```
antoyjchubekova@dk8n52 ~/work/arch-pc/lab07 $ touch lab7-3.asm
```

Рис. 4.15: РИС.16 Создание файла

```

lab7-3.asm      [----] 12 L:[ 1+ 2  3/
#include 'in_out.asm'

section .data
msg1 db "Наибольшее число: ",0h
A dd '81'
B dd '22'
C dd '72'

section .bss
min resb 10

section .text
global _start
_start:

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

mov ecx,[A]
mov [min],ecx

cmp ecx,[C]
jl check_B
mov ecx,[C]
mov [min],ecx

check_B:
mov eax,min
call atoi
mov [min],eax

mov ecx,[min]
cmp ecx,[B]
jl fin
mov ecx,[B]
mov [min],ecx

fin:
mov eax, msg1
call sprint
mov eax,[min]

```

Рис. 4.16: РИС.17 Редактирование файла

Создадим исполняемый файл и запустим его. Видим, что правильно работает и программа выдает число 22.(РИС.18)

```
antoyjchubekova@dk8n52 ~/work/arch-pc/lab07 $ nasm -f elf lab7-3.asm
antoyjchubekova@dk8n52 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-3 lab7-3.o
antoyjchubekova@dk8n52 ~/work/arch-pc/lab07 $ ./lab7-3
Наибольшее число: 22
```

Рис. 4.17: РИС.18 Запуск исполняемого файла

Создадим файл lab7-4.asm в котором будем работать.(РИС.19) Напишем программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции, указанная в варианте 14, в соответствии с условиями и выводит результат вычислений.(РИС.20). Сама программа вынлядит следующим образом:

```
%include 'in_out.asm'

section .data
msg DB 'Введите x:',0h
msg1 DB "Введите a:",0h
msg2: DB 'Ответ:',0h

section .bss
x: RESB 10
a: RESB 10
otv: RESB 10

section .text
global _start
_start:
```

```
    mov eax,msg  
call sprint
```

```
    mov ecx,x  
mov edx,10  
call sread
```

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

```
    mov eax,msg1  
call sprint
```

```
    mov ecx,a  
mov edx,10  
call sread
```

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

```
    mov ebx,3
```

```
    mov ecx,[x]  
cmp ecx,[a]  
jl inache  
mov eax, [x]  
mul ebx
```



```
inc eax
jmp chan

inache:
mov eax,[a]
mul ebx
inc eax

chan:
mov [otv],eax

fin:
mov eax,msg2
call sprint

mov eax,[otv]
call iprintLF

call quit.
```



```
antoyjchubekova@dk8n52 ~/work/arch-pc/lab07 $ touch lab7-4.asm
```

Рис. 4.18: РИС.19 Создание файла

```

lab7-4.asm      [----]  0 L:[  1+14  15/ 66]
#include 'in_out.asm'

section .data
msg DB 'Введите x: ',0h
msg1 DB "Введите a: ",0h
msg2: DB 'Ответ:',0h

section .bss
x: RESB 10
a: RESB 10
otv: RESB 10

section .text
global _start
_start:

mov eax,msg
call sprint

mov ecx,x
mov edx,10
call sread

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

mov eax,msg1
call sprint

mov ecx,a
mov edx,10
call sread

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

mov ebx,3

mov ecx,[x]
cmp ecx,[a]

```

Рис. 4.19: РИС.20 Редактирование файла

Создадим исполняемый файл и запустим его. Поставив с калькулятором видим что все правильно работает.(РИС.21) и (РИС.22)

```
antoychubekova@dk8n52 ~/work/arch-pc/lab07 $ nasm -f elf lab7-4.asm
antoychubekova@dk8n52 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-4 lab7-4.o
antoychubekova@dk8n52 ~/work/arch-pc/lab07 $ ./lab7-4
Введите x: 2
Введите a: 3
Ответ:10
```

Рис. 4.20: РИС.21 Запуск исполняемого файла

```
antoychubekova@dk8n52 ~/work/arch-pc/lab07 $ nasm -f elf lab7-4.asm
antoychubekova@dk8n52 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-4 lab7-4.o
antoychubekova@dk8n52 ~/work/arch-pc/lab07 $ ./lab7-4
Введите x: 4
Введите a: 2
Ответ:13
```

Рис. 4.21: РИС.22 Запуск исполняемого файла

## 5 Выводы

В ходе выполнения лабораторной работы №7 я научилась пользоваться командами условного и безусловного переходов. Также приобрела некоторые навыки написания программ и использованием переходов. Еще познакомилась с назначением и структурой файла листинга. Далее используя полученные навыки, я написала программу находящая наименьшую из трех целочисленных переменных (81,22,72). Вместе с тем написала программу которая для введенных с клавиатуры значений  $x$ , а вычисляет значения функций удовлетворяющее условию и выводит результат.

## **Список литературы**

-<https://esystem.rudn.ru/course/view.phpid=4975>