

# **Отчёт по лабораторной работе № 8**

**Команды безусловного и условного переходов в Nasm.  
Программирование ветвлений.**

Поляков Глеб Сергеевич

# Содержание

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Цель работы</b>                                      | <b>5</b>  |
| <b>2</b> | <b>Задание</b>  | <b>6</b>  |
| 2.1      | Задание для самостоятельной работы . . . . .            | 6         |
| <b>3</b> | <b>Теоретическое введение</b>                           | <b>7</b>  |
| 3.1      | 8.2. Теоретическое введение . . . . .                   | 7         |
| 3.1.1    | 8.2.1. Команды безусловного перехода . . . . .          | 7         |
| 3.1.2    | 8.2.2. Команды условного перехода . . . . .             | 8         |
| 3.2      | Таблица 8.2. Регистр флагов . . . . .                   | 8         |
| 3.2.1    | 8.2.2.2. Описание инструкции <code>str</code> . . . . . | 10        |
| 3.2.2    | 8.2.2.3. Описание команд условного перехода. . . . .    | 10        |
| 3.2.3    | Таблица 8.3. . . . .                                    | 11        |
| 3.3      | 8.2.3. Файл листинга и его структура . . . . .          | 14        |
| <b>4</b> | <b>Выполнение лабораторной работы</b>                   | <b>16</b> |
| 4.1      | Выполнение . . . . .                                    | 17        |
| <b>5</b> | <b>Выводы</b>   | <b>21</b> |
|          | <b>Список литературы</b>                                | <b>22</b> |

## Список иллюстраций

|     |  |    |
|-----|--|----|
| 4.1 | Текст программы lab8-3.asm . . . . .     | 17 |
| 4.2 | Результат программы lab8-3.asm . . . . . | 18 |
| 4.3 | Текст программы lab8-3.asm . . . . .     | 19 |
| 4.4 | Результат программы lab8-3.asm . . . . . | 20 |

## **Список таблиц**

# 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. Создайте каталог для программам лабораторной работы No 8, перейдите в него и создайте файл lab8-1.asm
2. Введите в файл lab8-1.asm текст программы из листинга 8.1.
3. Создайте файл lab8-2.asm в каталоге ~/work/arch-pc/lab08. Внимательно изучите текст программы из листинга 8.3 и введите в lab8-2.asm.
4. Создайте файл листинга для программы из файла lab8-2.asm. Внимательно ознакомьтесь с его форматом и содержимым. Подробно объяснить содержимое трёх строк файла листинга по выбору.

### 2.1 Задание для самостоятельной работы

1. Напишите программу нахождения наименьшей из 3 целочисленных переменных  $a$ ,  $b$  и  $c$ . Значения переменных выбрать из табл. 8.5 в соответствии с вариантом, полученным при выполнении лабораторной работы No 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 8.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы No 7. Создайте исполняемый файл и проверьте его работу для значений  $x$  и  $a$  из 8.6.

## 3 Теоретическое введение

### 3.1 8.2. 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.

#### 3.1.1 8.2.1. Команды безусловного перехода

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

```
jmp <адрес_перехода>
```

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 (см. табл. 8.1).

Таблица 8.1. Типы операндов инструкции `jmp`

| Тип операнда             | Описание  |
|--------------------------|---|
| <code>jmp label</code>   | переход на метку <code>label</code>                               |
| <code>jmp [label]</code> | переход по адресу в памяти, помеченному меткой <code>label</code> |
| <code>jmp eax</code>     | переход по адресу из регистра <code>eax</code>                    |

В следующем примере рассмотрим использование инструкции `jmp`:

```
label:
    ... ;
    ... ; команды
    ... ;
    jmp label
```

### 3.1.2 8.2.2. Команды условного перехода

Как отмечалось выше, для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов.

#### 8.2.2.1. Регистр 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. В следующей таблице указано положение битовых флагов в регистре флагов. Флаги состояния (биты 0, 2, 4, 6, 7 и 11) отражают результат выполнения арифметических инструкций, таких как `ADD`, `SUB`, `MUL`, `DIV`.

## 3.2 Таблица 8.2. Регистр флагов



| Бит | Обозначение | Название   | Описание   |
|-----|-------------|--|--|
| 0   | CF          | Carry Flag - Флаг переноса                           | Устанавливается в 1, если при выполнении предыдущей операции произошёл перенос из старшего бита или если требуется заём (при вычитании). Иначе установлен в 0. |
| 2   | PF          | Parity Flag - Флаг чётности                          | Устанавливается в 1, если младший байт результата предыдущей операции содержит чётное количество битов, равных 1.  |
| 4   | AF          | Auxiliary Carry Flag - Вспомогательный флаг переноса | Устанавливается в 1, если в результате предыдущей операции произошёл перенос (или заём) из третьего бита в четвёртый.  |
| 6   | ZF          | Zero Flag - Флаг нуля                                | Устанавливается 1, если результат предыдущей команды равен 0.  |
| 7   | SF          | Sign Flag - Флаг знака                               | Равен значению старшего значащего бита результата, который является знаковым битом в знаковой арифметике.  |

| Бит | Обозначение | Название                          | Описание  |
|-----|-------------|-----------------------------------|---|
| 11  | OF          | Overflow Flag - Флаг переполнения | Устанавливается в 1, если целочисленный результат слишком длинный для размещения в целевом операнде (регистре или ячейке памяти). |

Более подробную информацию о регистре флагов см. [14; 15].

### 3.2.1 8.2.2.2. Описание инструкции `cmp`

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:

`cmp <операнд_1>, <операнд_2>`

Команда `cmp`, так же как и команда вычитания, выполняет вычитание -, но результат вычитания никуда не записывается и единственным результатом команды сравнения является формирование флагов.

Примеры.

`cmp ax, '4'` ; сравнение регистра `ax` с символом `4`

`cmp ax, 4` ; сравнение регистра `ax` со значением `4`

`cmp al, cl;` ; сравнение регистров `al` и `cl`

`cmp [buf], ax` ; сравнение переменной `buf` с регистром `ax`

### 3.2.2 8.2.2.3. 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. В табл. 8.3. представлены команды условного перехода, которые обычно ставятся после команды сравнения стр. В их мнемониках 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.

### 3.2.3 Таблица 8.3.

| Типы операндов | Мнемокод | Критерий условия |       |   | Значения флагов | Комментарий           |
|----------------|----------|------------------|-------|---|-----------------|-----------------------|
|                |          | да               | а     | б |                 |                       |
| Любые          | JE       | a=b              | ZF=1  |   |                 | Переход если равно    |
| Любые          | JNE      | a≠b              | ZF=0  |   |                 | Переход если не равно |
| Со знаком      | JL/JNGE  | a<b              | SF≠OF |   |                 | Переход если меньше   |

| Типы операндов |         | Кри-<br>тен-<br>рий<br>услов-<br>но-<br>го<br>пе-<br>ре-<br>хо-<br>да | а                       | Значения флагов | Комментарий                   |
|----------------|---------|---|-------------------------|-----------------|-------------------------------|
| Со знаком      | JLE/JNG | $a \leq b$  | $SF \neq OF$ или $ZF=1$ |                 | Переход если меньше или равно |
| Со знаком      | JG/JNLE | $a > b$   | $SF=OF$ и $ZF=0$        |                 | Переход если больше           |
| Со знаком      | JGE/JNL | $a \geq b$  | $SF=OF$                 |                 | Переход если больше или равно |
| Без знака      | JB/JNAE | $a < b$   | $CF=1$                  |                 | Переход если ниже             |
| Без знака      | JBE/JNA | $a \leq b$  | $CF=1$ или $ZF=1$       |                 | Переход если ниже или равно   |
| Без знака      | JA/JNBE | $a > d$   | $CF=0$ и $ZF=0$         |                 | Переход если выше             |

|                |          |                 |                             |
|----------------|----------|-----------------|-----------------------------|
| Типы операндов | Мнемокод | Значения флагов | Комментарий                 |
|                |          |                 |                             |
| Без знака      | JAE/JNB  | a ≥ d CF=0      | Переход если выше или равно |

Примечание: термины «выше» («a» от англ. «above») и «ниже» («b» от англ. «below») применимы для сравнения беззнаковых величин (адресов), а термины «больше» («g» от англ. «greater») и «меньше» («l» от англ. «lower») используются при учёте знака числа. Таким образом, мнемонику инструкции JA/JNBE можно расшифровать как «jump if above (переход если выше) / jump if not below equal (переход если не меньше или равно)». Помимо перечисленных команд условного перехода существуют те, которые можно использовать после любых команд, меняющих значения флагов (табл. 8.4).

###Таблица 8.4. | Мнемокод | Значение флага для осуществления перехода | Мнемокод | Значение флага для осуществления перехода | |-----|-----|-----|-----|  
 || JZ | ZF = 1 | JNZ | ZF = 0 || JS | SF = 1 | JNS | SF = 0 || JC | CF = 1 | JNC | CF = 0 || JO | OF = 1 | JNO | OF = 0 || JP | PF = 1 | JNP | PF = 0 |

В качестве примера рассмотрим фрагмент программы, которая выполняет умножение переменных a и b и если произведение превосходит размер байта,

передает управление на метку Error.

```
mov    al, a
mov    bl, b
mul    bl
jc     Error
```

### 3.3 8.2.3. Файл листинга и его структура

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. Ниже приведён фрагмент файла листинга.

```
10 00000000 B804000000
11 00000005 BB01000000
12 0000000A B9[00000000]
13 0000000F BA0D000000
14
15 00000014 CD80
```

```
mov    eax,4
mov    ebx,1
mov    ecx,hello
mov    edx,helloLen
int    80h
```

Строки в первой части листинга имеют следующую структуру (рис. 8.2). Рис. 8.2. Структура листинга 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 им присваивается).

## 4 Выполнение лабораторной работы

### 1. Создал каталог и файл lab8-1.asm

```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg1: DB 'Сообщение №1',0
5 msg2: DB 'Сообщение №2',0
6 msg3: DB 'Сообщение №3',0
7
8 SECTION .text
9 GLOBAL _start
10 _start:
11
12 jmp _label3
13
14 _label1:
15 mov eax, msg1
16 call sprintf
17 jmp _end
18 _label2:
19 mov eax, msg2
20 call sprintf
21 jmp _label1
22 _label3:
23 mov eax, msg3
24 call sprintf
25 jmp _label2
26 _end:
27 call quit
```

### 2. Ввел текст программы в файл (рис. ??)

```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg1: DB "Message 1: ",0
5 msg2: DB "Performance ratio : ",0
6 A: DB "A de '28'"
7 C: DB "C de '58'"
8
9 SECTION .bss
10 resb 36
11 B: resb 10
12
13 SECTION .text
14 GLOBAL _start
15 _start:
16 mov eax, msg1
17 call sprintf
18
19 mov ecx, B
20 call streb
21
22 mov eax, B
23 call atoi
24 mov [A],eax
25
26 mov [max],0
27 mov [max],ecx
28
29 cmp ecx, C
30 jg check_B
31 mov ecx
32 mov [max],ecx
33
34 check_B:
35 mov eax, max
36 call atoi
37 mov [max],eax
38
39 mov ecx, [max]
40 cmp ecx, [B]
41 jg fin
42 mov ecx, [B]
43 mov [max],ecx
44 fin:
45 mov eax, msg2
46 call sprintf
47 mov eax, [max]
48 call sprintf
49 call quit
```

### 3. Создал файл lab8-2.asm, ввел текст программы (рис. ??)

4. 12 00000005 BB01000000 mov ebx,1 13 0000000A B9[00000000] mov ecx,hello  
14 0000000F BA0E000000 mov edx,helloLen

12 – Помещение значения BB01000000 в операнд 00000005 13 – Помеще-



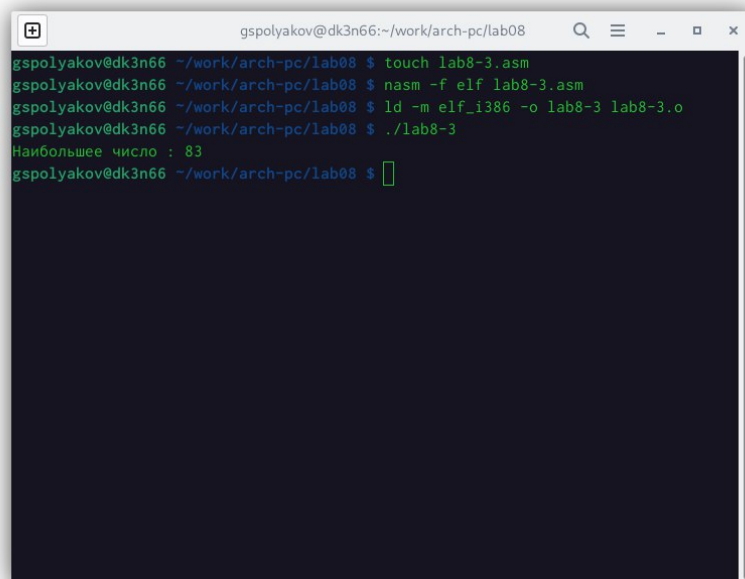
ние значения B9[00000000] в операнд 0000000A 14 – Помещение значения BA0E000000 в операнд 0000000F

## 4.1 Выполнение

1. Написал программу нахождения наименьшей из 3 целочисленных переменных a, b и c (рис. 4.1). Значения переменных выбрал из табл. 8.5 в соответствии с вариантом, полученным при выполнении лабораторной работы No 7. Создал исполняемый файл и проверил его работу. (рис. 4.2)

```
1  %include 'in_out.asm'
2
3  SECTION .data
4  msg1: DB 'Введите x: ',0
5  msg2: DB 'Введите a: ',0
6  msg3: DB 'Результат: ',0
7  A dd 4
8  X dd 3
9  B dd 5
10 SECTION .bss
11 max: resb 80
12 x: resb 80
13
14 SECTION .text
15 GLOBAL _start
16 _start:
17     mov eax,msg1
18     call sprint
19
20     mov ecx,x
21     mov edx,80
22     call sread
23
24     mov eax,x
25     call atoi
26     mov [X],eax
27
28     mov eax,msg2
29     call sprint
30
31     mov ecx,x
32     mov edx,80
33     call sread
34
35     mov eax,x
36     call atoi
37     mov [A],eax
38
39     mov eax,[A]
40     cmp eax,[X]
41     je _equal
42     jne _not_equal
43
44 _equal:
45     add eax,[X]
46     mov [max],eax
47     mov eax,msg3
48     call sprint
49     mov eax,[max]
50     call iprintLF
51     call quit
52
53
54 _not_equal:
55     mov ebx,[B]
56     mov eax,[X]
57     mul ebx
58     mov [max],eax
59     mov eax,msg3
60     call sprint
61     mov eax,[max]
```

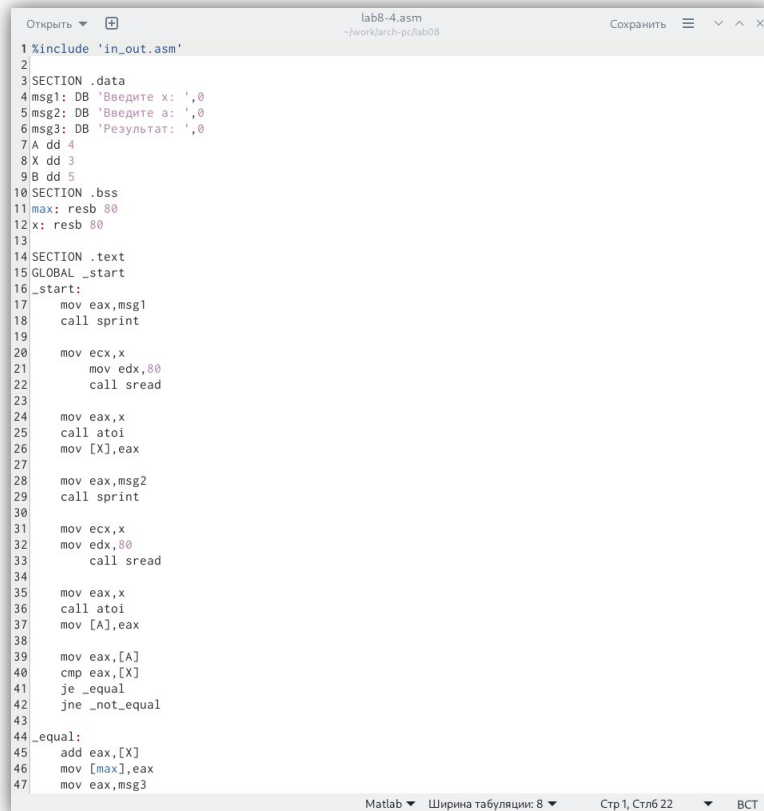
Рис. 4.1: Текст программы lab8-3.asm

A terminal window with a dark background and light green text. The window title is 'gspolyakov@dk3n66:~/work/arch-pc/lab08'. The terminal shows the following commands and output:

```
gspolyakov@dk3n66 ~/work/arch-pc/lab08 $ touch lab8-3.asm
gspolyakov@dk3n66 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
gspolyakov@dk3n66 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
gspolyakov@dk3n66 ~/work/arch-pc/lab08 $ ./lab8-3
Наибольшее число : 83
gspolyakov@dk3n66 ~/work/arch-pc/lab08 $
```

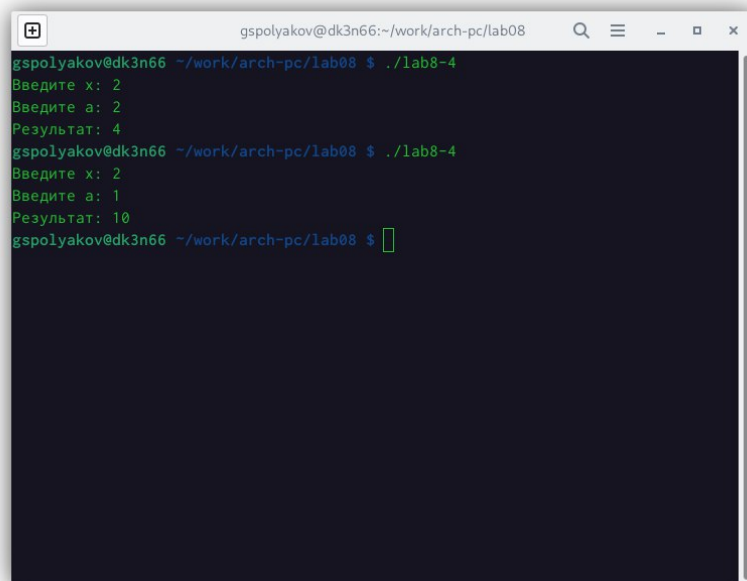
Рис. 4.2: Результат программы lab8-3.asm

2. Напишите программу, которая для введенных с клавиатуры значений  $x$  и  $a$  вычисляет значение заданной функции  $f(x)$  и выводит результат вычислений (рис. 4.3). Вид функции  $f(x)$  выбрать из таблицы 8.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы No 7. Создайте исполняемый файл и проверьте его работу для значений  $x$  и  $a$  из 8.6.(рис. 4.4)



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg1: DB 'Введите x: ',0
5 msg2: DB 'Введите a: ',0
6 msg3: DB 'Результат: ',0
7 A dd 4
8 X dd 3
9 B dd 5
10 SECTION .bss
11 max: resb 80
12 x: resb 80
13
14 SECTION .text
15 GLOBAL _start
16 _start:
17     mov eax,msg1
18     call sprint
19
20     mov ecx,x
21     mov edx,80
22     call sread
23
24     mov eax,x
25     call atoi
26     mov [X],eax
27
28     mov eax,msg2
29     call sprint
30
31     mov ecx,x
32     mov edx,80
33     call sread
34
35     mov eax,x
36     call atoi
37     mov [A],eax
38
39     mov eax,[A]
40     cmp eax,[X]
41     je _equal
42     jne _not_equal
43
44 _equal:
45     add eax,[X]
46     mov [max],eax
47     mov eax,msg3
```

Рис. 4.3: Текст программы lab8-3.asm



```
gspolyakov@dk3n66:~/work/arch-pc/lab08 $ ./lab8-4
Введите x: 2
Введите a: 2
Результат: 4
gspolyakov@dk3n66:~/work/arch-pc/lab08 $ ./lab8-4
Введите x: 2
Введите a: 1
Результат: 10
gspolyakov@dk3n66:~/work/arch-pc/lab08 $
```

Рис. 4.4: Результат программы lab8-3.asm

## 5 Выводы

Выполняя лабораторную работу, я изучил команды условного и безусловного переходов. Приобрел навыки написания программ с использованием переходов. Познакомился с назначением и структурой файла листинга.

## **Список литературы**