

# **Отчет по лабораторной работа №10**

**Группа НПИбд-02-22**

Стариков Данила Андреевич

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>3</b>
<b>2</b>	<b>Основная часть</b>	<b>4</b>
2.1	Выполнение лабораторной работы . . . . .	4
2.1.1	Реализация подпрограмм в NASM . . . . .	4
2.1.2	Отладка программ с помощью GDB . . . . .	5
2.2	Выполнение заданий для самостоятельной работы. . . . .	14
2.2.1	Задание 1. . . . .	14
2.2.2	Задание 2. . . . .	14
<b>3</b>	<b>Выводы</b>	<b>23</b>

# 1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

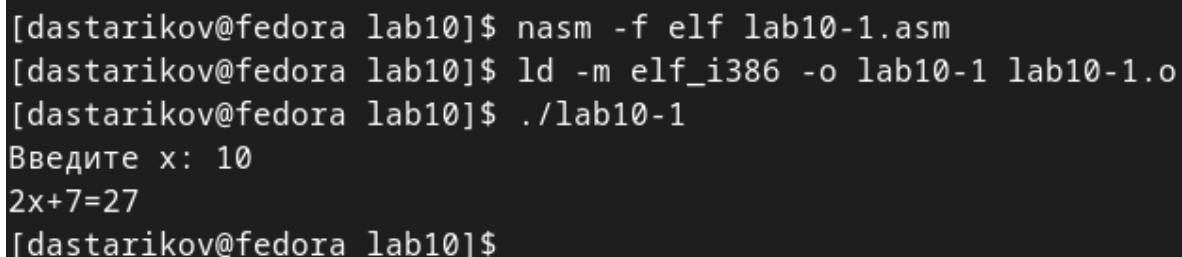
## 2 Основная часть

### 2.1 Выполнение лабораторной работы

#### 2.1.1 Реализация подпрограмм в NASM

Для выполнения лабораторной работы создали каталог `~/work/study/arch-rc/lab10`. В нем создали файл `lab10-1.asm` и ввели текст из Листинга 2.1. Также положили в каталог файл `in_out.asm`, использованный в лабораторной работе №6.

Создали исполняемый файл и запустили его (Рис. 2.1).



```
[dastarikov@fedora lab10]$ nasm -f elf lab10-1.asm
[dastarikov@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[dastarikov@fedora lab10]$ ./lab10-1
Введите x: 10
2x+7=27
[dastarikov@fedora lab10]$
```

Рис. 2.1: Результат запуска файла `lab10-1`.

Затем изменили текст программы, добавив еще одну подпрограмму `_subcalcul`, которая вычисляет значение  $g(x) = 3 * x - 1$ , а в результате вычисляется значение выражения  $f(g(x))$  и выводится на экран (Листинг 2.2).

Создали исполняемый файл и запустили его (Рис. 2.2).

```
[dastarikov@fedora lab10]$ nasm -f elf lab10-1.asm
[dastarikov@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[dastarikov@fedora lab10]$ ./lab10-1
Введите x: 3
f(x)=2x+7, g(x)= 3x-1, f(g(x)) = 23
[dastarikov@fedora lab10]$
```

Рис. 2.2: Вычисление  $f(g(x))$ .

### 2.1.2 Отладка программ с помощью GDB

Создали файл lab10-2.asm и ввели текст из Листинга 2.3.

Для работы с отладчиком GDB при трансляции исполняемого файла добавили ключ -g (Рис. 2.3).

```
[dastarikov@fedora lab10]$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
[dastarikov@fedora lab10]$ ld -m elf_i386 -o lab10-2 lab10-2.o
```

Рис. 2.3: Компиляция файла lab10-2.asm

Загрузили исполняемый файл в отладчик GDB и проверили работы по команде run (Рис. 2.4).

```
(gdb) run
Starting program: /home/dastarikov/work/study/arch-pc/lab10/lab10-2
Hello, world!
```

Рис. 2.4: Обычный запуск программы.

Далее установили брейкпоинт на метку `_start`, и еще раз запустили ее (Рис. 2.5).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 12.
(gdb) run
Starting program: /home/dastarikov/work/study/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:12
12      mov eax, 4
(gdb) |
```

Рис. 2.5: Установлен брейкпоинт.

Посмотрели дисассемблированный код программы в режиме AT&T и Intel (Рис. 2.6 и 2.7). Основные различия синтаксисов: противоположное расположение операнда-источника и операнда-приемника (в Intel - приемник, источник, а в AT&T - источник, приемник); в AT&T регистры пишутся после '%', а непосредственные операнды после '\$', в синтаксисе Intel операнды никак не помечаются.

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     $0x4,%eax
    0x08049005 <+5>:      mov     $0x1,%ebx
    0x0804900a <+10>:     mov     $0x804a000,%ecx
    0x0804900f <+15>:     mov     $0x8,%edx
    0x08049014 <+20>:     int     $0x80
    0x08049016 <+22>:     mov     $0x4,%eax
    0x0804901b <+27>:     mov     $0x1,%ebx
    0x08049020 <+32>:     mov     $0x804a008,%ecx
    0x08049025 <+37>:     mov     $0x7,%edx
    0x0804902a <+42>:     int     $0x80
    0x0804902c <+44>:     mov     $0x1,%eax
    0x08049031 <+49>:     mov     $0x0,%ebx
    0x08049036 <+54>:     int     $0x80
End of assembler dump.
(gdb) |
```

Рис. 2.6: Дисассемблированный код программы в режиме АТТ

```
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     eax,0x4
    0x08049005 <+5>:      mov     ebx,0x1
    0x0804900a <+10>:     mov     ecx,0x804a000
    0x0804900f <+15>:     mov     edx,0x8
    0x08049014 <+20>:     int     0x80
    0x08049016 <+22>:     mov     eax,0x4
    0x0804901b <+27>:     mov     ebx,0x1
    0x08049020 <+32>:     mov     ecx,0x804a008
    0x08049025 <+37>:     mov     edx,0x7
    0x0804902a <+42>:     int     0x80
    0x0804902c <+44>:     mov     eax,0x1
    0x08049031 <+49>:     mov     ebx,0x0
    0x08049036 <+54>:     int     0x80
End of assembler dump.
(gdb) |
```

Рис. 2.7: Дисассемблированный код программы в режиме Intel

Включили режим псевдографики для более удобного анализа (Рис. 2.8).



```
Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffd160 0xffffd160  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049000 0x8049000 <_start>  eflags   0x202    [ IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       0x0      0

B+> 0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov ebx,0x1
0x804900a <_start+10> mov ecx,0x804a000
0x804900f <_start+15> mov edx,0x8
0x8049014 <_start+20> int 0x80
0x8049016 <_start+22> mov eax,0x4
0x804901b <_start+27> mov ebx,0x1
0x8049020 <_start+32> mov ecx,0x804a008
0x8049025 <_start+37> mov edx,0x7
0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax,0x1

native process 4901 In: _start L12 PC: 0x8049000
(gdb) layout regs
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/dastarikov/work/study/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:12
(gdb) |
```

Рис. 2.8: Режим псевдографики GDB.

Проверили установленные брейкпоинты по команде `info breakpoints` (Рис. 2.9).

```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffd160 0xffffd160  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049000 0x8049000 <_start>  eflags    0x202    [ IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       0x0      0

B+> 0x8049000 <_start> mov    eax,0x4
0x8049005 <_start+5> mov    ebx,0x1
0x804900a <_start+10> mov    ecx,0x804a000
0x804900f <_start+15> mov    edx,0x8
0x8049014 <_start+20> int     0x80
0x8049016 <_start+22> mov    eax,0x4
0x804901b <_start+27> mov    ebx,0x1
0x8049020 <_start+32> mov    ecx,0x804a008
0x8049025 <_start+37> mov    edx,0x7

native process 4901 In: _start L12 PC: 0x8049000
(gdb) layout regs
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/dastarikov/work/study/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:12
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint       keep y   0x08049000 lab10-2.asm:12
      breakpoint already hit 1 time
(gdb) |

```

Рис. 2.9: Список установленных брейкпоинтов.

Добавили еще одну точку остановки по адресу инструкции `mov ebx, 0x0` (Рис. 2.10).

```

dastarikov@fedora:~/work/study/arch-pc/lab10 — /usr/bin/mc -P /var/tmp/mc-dastarikov/mc.pwd.3009
[ Register Values Unavailable ]

0x804901b <_start+27> mov    ebx,0x1
0x8049020 <_start+32> mov    ecx,0x804a008
0x8049025 <_start+37> mov    edx,0x7
0x804902a <_start+42> int     0x80
0x804902c <_start+44> mov    eax,0x1
b+ 0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int     0x80

exec No process in: L?? PC: ??
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 12.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
(gdb) break 0x8049031
Function "0x8049031" not defined.
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab10-2.asm, line 25.
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint       keep y   0x8049000 lab10-2.asm:12
2     breakpoint       keep y   0x8049031 lab10-2.asm:25
(gdb)

```

Рис. 2.10: Добавление новой точки остановки.

С помощью инструкции `stepi` (сокращенно `si`) выполнили 5 последующих инструкций. В результате изменяются значения регистров `eax`, `ebx`, `ecx`, `edx` (Рис. 2.11).

```

dastarikov@fedora: ~/work/study/arch-pc/lab10 — /usr/bin/mc -P /var/tmp/mc-dastarikov/mc.pwd.3009
Register group: general
eax      0x8      8      ecx      0x804a000    134520832
edx      0x8      8      ebx      0x1          1
esp      0xffffd160 0xffffd160  ebp      0x0          0x0
esi      0x0      0      edi      0x0          0
eip      0x8049016 0x8049016 <_start+22>  eflags    0x202        [ IF ]
cs       0x23     35     ss       0x2b         43
ds       0x2b     43     es       0x2b         43
fs       0x0      0      gs       0x0          0

0x8049008 <_start>    mov     eax,0x4
0x8049005 <_start+5>   mov     ebx,0x1
0x804900a <_start+10>  mov     ecx,0x804a000
0x804900f <_start+15>  mov     edx,0x8
0x8049014 <_start+20>  int     0x80
> 0x8049016 <_start+22> mov     eax,0x4
0x804901b <_start+27>   mov     ebx,0x1
0x8049020 <_start+32>   mov     ecx,0x804a008
0x8049025 <_start+37>   mov     edx,0x7
0x804902a <_start+42>   int     0x80

native process 5380 In: _start L18 PC: 0x8049016

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) nDebuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

Breakpoint 1, _start () at lab10-2.asm:12
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.11: Выполнение первых 5 инструкций.

Посмотрели значения переменных `msg1` и `msg2`, причем адрес памяти `msg1` задали по имени переменной, а адрес памяти `msg2` вписали вручную, взяв его из дисассемблированного кода программы (Рис. 2.12).

```

(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 2.12: Вывод значения переменных `msg1` и `msg2`.

С помощью команды `set` заменили в `msg1` первый символ, в переменной `msg2`

- второй (Рис. 2.13 и 2.14).

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb)
```

Рис. 2.13: Изменение значения переменной msg1.

```
(gdb) set {char}0x804a009 = '0'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "wOrld!\n\034"
(gdb) |
```

Рис. 2.14: Изменение значения переменной msg2.

Вывели значения регистра `edx` в десятичном, шестнадцатичном и двоичном формате, соответственно (Рис. 2.15).

```
(gdb) p/s $edx
$1 = 8
(gdb) p/x $edx
$2 = 0x8
(gdb) p/t $edx
$3 = 1000
(gdb) |
```

Рис. 2.15: Значение регистра `edx`.

Используя команду `set` изменили значение регистра `ebx` сначала на символ '2', а затем на число 2, и сравнили вывод значения регистра в десятичном формате.

В результате присвоения регистра значение символа '2', выводится число 50, что соответствует символу в '2' в таблице ASCII (Рис. 2.16).

```
(gdb) set $ebx = '2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx = 2
(gdb) p/s $ebx
$5 = 2
(gdb)
```

Рис. 2.16: Изменение значения регистра ebx.

Затем завершили выполнение программы командой `continue` (сокращенно `c`), и вышли из GDB по команде `exit`.

Для примера обработки аргументов командной строки в GDB скопировали файл `lab9-2.asm` из Лабораторной работы №9 (программа выводит на экран аргументы командной строки) в файл `lab10-3.asm`, создали исполняемый файл и запустили в GDB с ключом `--args` (Рис. 2.17).

```
[dastarikov@fedora lab10]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
[dastarikov@fedora lab10]$ ld -m elf_i386 -o lab10-3 lab10-3.o
[dastarikov@fedora lab10]$ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
```

Рис. 2.17: Загрузка в GDB программы `lab10-3` с аргументами.

Установили брейкпоинт в начале программы, посмотрели значение регистра `esp`, равное количеству аргументов командной строки, включая имя программы, и остальные позиции стека (Рис. 2.18). Каждый элемент стека занимает 4 байта, поэтому для получения следующего элемента стека мы добавляем 4 к адресу вершины.

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 12.
(gdb) run
Starting program: /home/dastarikov/work/study/arch-pc/lab10/lab10-3 аргумент1 аргумент 2 аргумент\ 3
Breakpoint 1, _start () at lab10-3.asm:12
12      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 2.18: Значения элементов стека.

## 2.2 Выполнение заданий для самостоятельной работы.

### 2.2.1 Задание 1.

Написали программу lab10-4.asm (Листинг 2.4), являющуюся измененной версией программы lab09-var12.asm, но вычисление  $f(x) = 15 * x - 9$  реализовано как подпрограмма (Рис. 2.19).

```
[dastarikov@fedora lab10]$ nasm -f elf lab10-4.asm
[dastarikov@fedora lab10]$ ld -m elf_i386 -o lab10-4 lab10-4.o
[dastarikov@fedora lab10]$ ./lab10-4 1 2 3 4
Функция: f(x)=15*x-9
Результат: 114
[dastarikov@fedora lab10]$
```

Рис. 2.19: Результат работы программы lab10-4.

### 2.2.2 Задание 2.

Создали программу lab10-5.asm по Листингу 2.5, вычисляющую выражение  $(3 + 2) * 4 + 5$ , но неверно (Рис. 2.20).

```
[dastarikov@fedora lab10]$ nasm -f elf -g -l lab10-5.lst lab10-5.asm
[dastarikov@fedora lab10]$ ld -m elf_i386 -o lab10-5 lab10-5.o
[dastarikov@fedora lab10]$ ./lab10-5
Результат: 10
[dastarikov@fedora lab10]$
```

Рис. 2.20: Результат работы программы lab10-5 (неверный ответ).

Проанализировали с помощью отладчика GDB изменение значений регистров, чтобы найти ошибку. Установили брейкпоинт в `_start`, включили режим псевдографики и начали поочередно выполнять команды и следить за значениями регистров. Первая ошибка - результат  $3 + 2 = 5$  записан в регистре `ebx` (команда `add ebx, eax`), но команда `mul` всегда перемножает значение регистра `eax` с указанным сомножителем, поэтому `mul ecx` дает неверный результат:  $(3+2)*4 = 5$ . (Рис. 2.21). Затем на убеждении, что регистр `ebx` содержит корректный результат вычислений, к нему добавляется число 5, результат запоминается в регистре `edi`, а затем выводится (Рис. 2.22).

```

Register group: General
eax      0x5      8      ecx      0x4      4      edx      0x0      0
ebx      0x5      5      esp      0xffffd1a0 4      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0      eip      0x80490fb 0x80490fb <_start+19>
eflags   0x206    [ PF IF ]  cs      0x23     35      ss      0x2b     43
ds       0x2b     43      es      0x2b     43      fs      0x0      0
gs       0x0      0

0x80490e0 <_start> mov ebx,0x3
0x80490e1 <_start+5> mov eax,0x2
0x80490f2 <_start+10> add ebx,eax
0x80490f4 <_start+12> mov ecx,0x4
0x80490f5 <_start+17> mul ecx
> 0x80490fb <_start+19> add ebx,0x5
0x8049100 <_start+22> mov edi,ebx
0x8049101 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8049086 <printf>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 add BYTE PTR [eax],al

native process 5150 In: _start
Starting program: /home/dastarikov/work/study/arch-pc/lab10/lab10-5
This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) nDebuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

Breakpoint 1, _start () at lab10-5.asm:12
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) |

```

Рис. 2.21: Некорректное использование команды `mul`.

```
Register group: general
eax      0x0      8      ecx      0x4      4      cdx      0x0      0
ebx      0xa      10     esp      0xffffd1a0 4      ebp      0x0      0x0
esi      0x0      0      edi      0xa      10     eip      0x8049100 0x8049100 <_start+24>
eflags   0x206    [ PF IF ]  cs      0x23     35     ss      0x2b     43
ds       0x2b     43     es      0x2b     43     fs      0x0      0
gs       0x0      0

0x80490c8 <_start> mov ebx,0x3
0x80490cd <_start+5> mov eax,0x2
0x80490f2 <_start+10> add ebx,eax
0x80490f4 <_start+12> mov ecx,0x4
0x80490f9 <_start+17> mul ecx
0x80490fb <_start+19> add ebx,0x5
0x8049102 <_start+22> mov edi,ebx
> 0x8049100 <_start+24> mov eax,0x804a000
0x8049109 <_start+29> call 0x8049000 <_start+24> <sprintf>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8049000 <_start+24> <printf>
0x8049111 <_start+41> call 0x8049000 <_start+24> <printf>
0x8049116 add BYTE PTR [eax],al

native process 5434 In: _start L21 PC: 0x8049100
This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) nDebuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

Breakpoint 1, _start () at lab10-5.asm:12
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рис. 2.22: В регистре ebx содержится неверный ответ.

Исправили текст программы (Листинг [-lst:lst06]) и убедились в верном вычислении результата (Рис [-fig:fig24]).

```
[dastarikov@fedora lab10]$ nasm -f elf -g -l lab10-5.lst lab10-5.asm
[dastarikov@fedora lab10]$ ld -m elf_i386 -o lab10-5 lab10-5.o
[dastarikov@fedora lab10]$ ./lab10-5
Результат: 25
[dastarikov@fedora lab10]$
```

Рис. 2.23: Правильный результат вычисления выражения.

Созданные файлы \*.asm скопировали в каталог ~/work/study/2022-2023/“Архитектура компьютера”/archpc/labs/lab10/ и загрузили на Github.



---

**Листинг 2.1 .** Пример программы с использованием вызова подпрограммы

---

```
%include 'in_out.asm'

SECTION .data
    msg: DB 'Введите x: ',0
    result: DB '2x+7=',0

SECTION .bss
    x: RESB 80
    rez: RESB 80

SECTION .text

GLOBAL _start
_start:

;-----
; Основная программа
;-----

    mov eax, msg
    call sprint

    mov ecx, x
    mov edx, 80
    call sread

    mov eax, x
    call atoi

    call _calcul ; Вызов подпрограммы _calcul

    mov eax, result
    call sprint
    mov eax, [rez]
    call iprintLF

    call quit

;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [rez], eax
    ret ; выход из подпрограммы
```

---

**Листинг 2.2** Вычисление выражения  $f(g(x))$  с использованием жвух подпрограмм.

---

```
%include 'in_out.asm'

SECTION .data
    msg: DB 'Введите x: ',0
    result: DB 'f(x)=2x+7, g(x)= 3x-1, f(g(x)) = ',0
```

```
SECTION .bss
    x: RESB 80
    rez: RESB 80
```

```
SECTION .text
```

```
GLOBAL _start
_start:
```

```
;-----
; Основная программа
;-----

    mov eax, msg
    call sprint

    mov ecx, x
    mov edx, 80
    call sread

    mov eax, x
    call atoi

    call _calcul ; Вызов подпрограммы _calcul

    mov eax, result
    call sprint
    mov eax, [rez]
    call iprintLF
    call quit

;-----
; Подпрограмма вычисления
; выражения "f(g(x))"
_calcul:
    call _subcalcul ; сначала вычисляем g(x)
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [rez], eax
    ret ; выход из подпрограммы
```

```
;-----
; Подпрограмма вычисления
; выражения "g(x)=3x-1"
```

```
_subcalcul:
```

```
    mov ebx, 3
```

```
    mul ebx
```

---

### Листинг 2.3 Программа вывода сообщения Hello world!

---

#### SECTION .data

```
msg1:    db "Hello, ",0x0
msg1Len:  equ $ - msg1
msg2:    db "world!",0xa
msg2Len:  equ $ - msg2
```

#### SECTION .text

```
global _start
```

```
_start:
```

```
    mov eax, 4
    mov ebx, 1
    mov ecx, msg1
    mov edx, msg1Len
    int 0x80
```

```
    mov eax, 4
    mov ebx, 1
    mov ecx, msg2
    mov edx, msg2Len
    int 0x80
```

```
    mov eax, 1
    mov ebx, 0
    int 0x80
```

---

---

## Листинг 2.4 Программа находит сумму значений функции

---

```
%include 'in_out.asm'

SECTION .data
    msg1 db "Функция: f(x)=15*x-9 ",0
    msg2 db "Результат: ",0
SECTION .text
    global _start

_start:

    mov eax, msg1
    call sprintLF

    pop ecx ; Извлекаем из стека в `ecx` количество
             ; аргументов (первое значение в стеке)
    pop edx ; Извлекаем из стека в `edx` имя программы
             ; (второе значение в стеке)
    sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
             ; аргументов без названия программы)
    mov esi, 0 ; Используем `esi` для хранения
             ; промежуточных сумм
next:
    cmp ecx,0h ; проверяем, есть ли еще аргументы
    jz _end ; если аргументов нет выходим из цикла
             ; (переход на метку `_end`)
    pop eax ; иначе извлекаем следующий аргумент из стека
    call atoi ; преобразуем символ в число
    call _calcul ; вызываем подпрограмму для вычисления f(x)

    add esi,eax ; добавляем к промежуточной сумме
             ; след. аргумент `esi=esi+eax`
    loop next ; переход к обработке следующего аргумента
_end:
    mov eax, msg2 ; вывод сообщения "Результат: "
    call sprint
    mov eax, esi ; записываем сумму в регистр `eax`
    call iprintLF ; печать результата
    call quit ; завершение программы

_calcul: ; подпрограмма для вычисления f(x)=15*x-9
    mov ebx, 15
    mul ebx
    sub eax, 9
    ret
```

---

**Листинг 2.5** Программа вычисления выражения  $(3+2)*4+5$  (неверная).

---

```
%include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start

_start:

    ; ---- Вычисление выражения (3+2)*4+5
    mov ebx,3
    mov eax,2
    add ebx,eax
    mov ecx,4
    mul ecx
    add ebx,5
    mov edi,ebx

    ; ---- Вывод результата на экран
    mov eax,div
    call sprint
    mov eax,edi
    call iprintLF

    call quit
```

---

---

**Листинг 2.6** Программа вычисления выражения  $(3+2)*4+5$  (неверная).

---

```
%include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start

_start:

    ; ---- Вычисление выражения (3+2)*4+5
    mov ebx,3
    mov eax,2
    add eax,ebx
    mov ecx,4
    mul ecx
    add eax,5
    mov edi,eax

    ; ---- Вывод результата на экран
    mov eax,div
    call sprint
    mov eax,edi
    call iprintLF

    call quit
```

---

## **3 Выводы**

В рамках лабораторной работы получили практические навыки использования подпрограмм и познакомились с методами отладки для более глубокого анализа работы программы.