

# **Отчёт по лабораторной работе №10**

Борунов Семён

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>3</b>	<b>Задания для самостоятельной работы</b>	<b>15</b>
<b>4</b>	<b>Выводы</b>	<b>18</b>

# Список иллюстраций

2.1	- . . . . .	7
2.2	$f(g(x))$ . . . . .	8
2.3	- . . . . .	8
2.4	- . . . . .	8
2.5	- . . . . .	9
2.6	- . . . . .	9
2.7	- . . . . .	9
2.8	- . . . . .	10
2.9	- . . . . .	10
2.10	- . . . . .	11
2.11	- . . . . .	11
2.12	- . . . . .	11
2.13	- . . . . .	11
2.14	- . . . . .	12
2.15	- . . . . .	12
2.16	- . . . . .	12
2.17	- . . . . .	12
2.18	- . . . . .	13
2.19	- . . . . .	13
2.20	- . . . . .	14
3.1	- . . . . .	15
3.2	- . . . . .	15
3.3	- . . . . .	16
3.4	- . . . . .	16
3.5	- . . . . .	17
3.6	- . . . . .	17

## Список таблиц

# 1 Цель работы

Освоить работу с подпрограммами и отладчиком gdb.

## 2 Выполнение лабораторной работы

Создадим рабочую директорию и файл. Запишем туда программу из листинга, исправив опечатки. (рис. 2.1)

```
Файл  Правка  Вид  Терминал  Выход  Стрелки
%include "in_out.asm"

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

SECTION .bss
x: RESB 80
res: RESB 80

SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprint

mov ecx, x
mov edx, 80
call sread

mov eax, x
call atoi

call _calcul

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

call quit

_calcul:
mov ebx, 2
mul ebx

add eax, 7
mov [res], eax

ret
```

Рис. 2.1: -

напишем программу, имитирующую сложную функцию. Функции назовем `_calcul` и `subcalcul`(рис. 2.2)

```
_calcul:
push ebx
mov ebx, 2
mul ebx

add eax, 7

pop ebx
ret

_subcalcul:
push ebx
mov ebx, 3
mul ebx
dec ebx
mov [res], eax
```

Рис. 2.2:  $f(g(x))$

Проверим ее работу (рис. [fig. 2.3])

```
ssborunov@dk6n51 ~/work/arc-pc/lab10 $ vim lab10-1.asm
ssborunov@dk6n51 ~/work/arc-pc/lab10 $ nasm -f elf lab10-1.asm && ld -m elf_i386 -o lab10-1 lab10-1.o && ./lab10-1
Введите x: 1
f(g(x))=3
ssborunov@dk6n51 ~/work/arc-pc/lab10 $
```

Рис. 2.3: -

Создадим файл `lab10-2.asm` и посмотрим, как она работает. Так же проасSEMBлируем его с другими ключами, чтобы была возможность открыть этот файл через `gdb`. (рис. 2.4)

```
ssborunov@dk6n51 ~/work/arc-pc/lab10 $ nasm -f elf lab10-2.asm && ld -m elf_i386 -o lab10-2 lab10-2.o && ./lab10-2
Hello, world!
ssborunov@dk6n51 ~/work/arc-pc/lab10 $ nasm -f elf -g -l lab10-2.lst && ld -m elf_i386 -o lab10-2 lab10-2.o
nasm: fatal: no input file specified
Type nasm -h for help.
ssborunov@dk6n51 ~/work/arc-pc/lab10 $ nasm -f elf -g -l lab10-2.lst lab10-2.asm && ld -m elf_i386 -o lab10-2 lab10-2.o
```

Рис. 2.4: -

Откроем `lab10-2` с помощью `gdb`. Запустим ее там(рис. 2.5)



```

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab10-2...
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/s/ssborunov/work/arc-pc/lab10/lab10-2
Hello, world!
[Inferior 1 (process 3518) exited normally]
(gdb) █

```

Рис. 2.5: -

Поставим точку останова (breakpoint) на метке `_start`. Посмотрим дизассемблированный код, начиная с этой метки. (рис. 2.6)

```

(gdb) disassemble _start
Undefined command: "disassemble". Try "help".
(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.

```

Рис. 2.6: -

Так же посмотрим как выглядит дизассемблированный код с синтаксисом Intel (рис. 2.7)

```

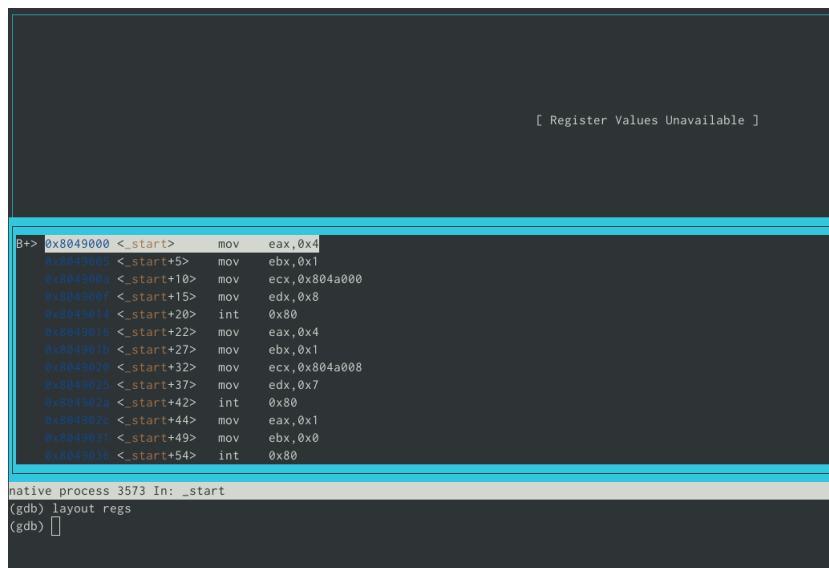
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:    mov     eax,0x4
      0x08049005 <+5>:    mov     ebx,0x1
      0x0804900a <+10>:   mov     ecx,0x804a000
      0x0804900f <+15>:   mov     edx,0x8
      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.

```

Рис. 2.7: -

В представлении АТТ в виде 16-ричного числа записаны первые аргументы всех команд, а в представлении intel так записываются адреса вторых аргументов.

включим режим псевдографики, с помощью которого отображается код программы и содержимое регистров(рис. 2.8)



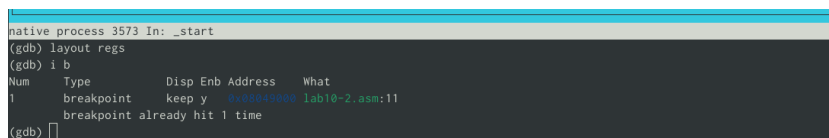
The screenshot shows a debugger window with a dark background. At the top, it says "[ Register Values Unavailable ]". Below that, there is a list of assembly instructions with their addresses and operands. The instructions are:

Address	Instruction
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
0x8049019	<_start+25> mov eax,0x4
0x804901e	<_start+32> mov ebx,0x1
0x8049023	<_start+37> mov ecx,0x804a008
0x8049028	<_start+42> mov edx,0x7
0x804902d	<_start+47> int 0x80
0x8049032	<_start+54> mov eax,0x1
0x8049037	<_start+59> mov ebx,0x0
0x804903c	<_start+66> int 0x80

Below the instructions, it says "native process 3573 In: \_start". Then, there are two lines of text: "(gdb) layout regs" and "(gdb) [ ]".

Рис. 2.8: -

Посмотрим информацию о наших точках останова. Сделать это можно коротко командой `i b` (рис. 2.9)



The screenshot shows a debugger window with a dark background. It starts with "native process 3573 In: \_start". Then, there are two lines of text: "(gdb) layout regs" and "(gdb) i b". Below that, there is a table showing breakpoint information:

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep y		0x8049000	lab10-2.asm:11

Below the table, it says "breakpoint already hit 1 time". Then, there are two lines of text: "(gdb) [ ]".

Рис. 2.9: -

добавим еще одну точку останова, но сделаем это по адресу (рис. 2.10)

```

B> 0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov ebx,0x1
0x804900a <_start+10> mov ecx,0x804a000
0x804900f <_start+15> mov edx,0x8
b+ 0x8049014 <_start+20> int 0x80
0x804901b <_start+22> mov eax,0x4
0x804901b <_start+27> mov ebx,0x1
0x804901d <_start+32> mov ecx,0x804a008
0x8049022 <_start+37> mov edx,0x7
0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax,0x1
0x8049031 <_start+49> mov ebx,0x0
0x804903c <_start+54> int 0x80

native process 3784 In: _start
breakpoint already hit 1 time
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /afs/.dk.sci.pfu.edu.ru/home/s/s/ssborunov/work/arc-pc/lab10/lab10-2
Breakpoint 1, _start () at lab10-2.asm:11
(gdb) b *0x8049014
Breakpoint 2 at 0x8049014: file lab10-2.asm, line 15.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x8049014 lab10-2.asm:11
breakpoint already hit 1 time
2 breakpoint keep y 0x8049014 lab10-2.asm:15
(gdb)

```

Рис. 2.10: -

Так же можно выводить значения регистров. Делается это командой `i r`. Псевдографика предствалена на (рис. 2.11)

```

eax 0x0 0 eax 0x0 0 eip 0x0 0
edx 0x0 0 ebx 0xffffc40 0 eip 0x0 0
esi 0x0 0 edi 0x0 0 eip 0x0 0
eflags 0x282 0 [ if ] cs 0x23 35 ss 0x10 43
ds 0x1b 43 es 0x1b 43 fs 0x0 0
gs 0x0 0

```

Рис. 2.11: -

В отладчике можно вывести текущее значение переменных. Сделать это можно например по имени (рис. 2.12) или по адресу (рис. 2.13)

```

Breakpoint 3, _start () at lab10-2.asm:11
11      mov eax, 4
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb)

```

Рис. 2.12: -

```

0x804a008: <error: Cannot access memory at address 0x804a008>
(gdb) si
(gdb) si
(gdb) si
(gdb) x/1sb 0x804a000
0x804a000 <msg1>: "Hello, "
(gdb)

```

Рис. 2.13: -

Так же отладчик позволяет менять значения переменных прямо во время выполнения программы (рис. 2.14)

```
11      mov     eax, 4
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
```

Рис. 2.14: -

Здесь тоже можно обращаться по адресам переменных(рис. 2.15). здесь был заменен первый символ переменной msg2 на символ отступа.

```
0x804a000 <msg1>: "hello, "
(gdb) set {char}&msg2=9
(gdb) x/1sb &msg2
0x804a008 <msg2>: "\torld!\n\034"
```

Рис. 2.15: -

Выводить можно так же содержимое регистров. Выведем значение edx в разных форматах: строчном, 16-ричном, двоичном(рис. 2.16)

```
0x804a008 <msg2>: "\torld!\n\034"
(gdb) p/s $edx
$1 = 0
(gdb) p/x
$2 = 0x0
(gdb) p/t
$3 = 0
```

Рис. 2.16: -

Как и переменным, регистрам можно задавать значения.(рис. 2.17)

```
(gdb) set $ebx="2"
evaluation of this expression requires the program to have a function "malloc".
(gdb) set $ebx=2
(gdb) p/s $ebx
$4 = 2
```

Рис. 2.17: -

Однако при попытке задать строчное значение, происходит ошибка.  
Завершим работу в gdb командами continue, она закончит выполнение программы, и exit, она завершит сеанс gdb.

Скопируем файл из лабораторной 9, переименуем и создадим исполняемый файл. Откроем отладчик и зададим аргументы. Создадим точку останова на метке `_start` и запустим программу(рис. 2.18)

```
ssborunov@ssborunov-VirtualBox:~/work/study/2022-2023/Arch-pc/study_2022-2023_a
rh-pc/labs/lab10$ ld -m elf_i386 -o lab10-3 lab10-3.o
ssborunov@ssborunov-VirtualBox:~/work/study/2022-2023/Arch-pc/study_2022-2023_a
rh-pc/labs/lab10$ gdb --args lab10-3 arg1 arg 2 "arg3"
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab10-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 8.
(gdb) run
Starting program: /home/ssborunov/work/study/2022-2023/Arch-pc/study_2022-2023_
arh-pc/labs/lab10/lab10-3 arg1 arg 2 arg3

Breakpoint 1, _start () at lab10-3.asm:8
8      pop ecx
(gdb) |
```

Рис. 2.18: -

Посмотрим на содержимое того, что расположено по адресу, находящемуся в регистре `esp` (рис. 2.19)

```
0      pop ecx
(gdb) x/x $esp
0xffffd090: 0x00000005
(gdb) |
```

Рис. 2.19: -

Далее посмотрим на все остальные аргументы в стеке. Их адреса располагаются в 4 байтах друг от друга(именно столько занимает элемент стека) (рис. 2.20)

```

8      pop ecx
(gdb) x/x $esp
0xffffd090: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd265: "/home/ssborunov/work/study/2022-2023/Arch-pc/study_2022-2023_a
rh-pc/labs/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd2bc: "arg1"
(gdb) x/s *(void**)(esp + 12)
0xffffd2c1: "arg"
(gdb) x/s *(void**)(esp + 20)
0xffffd2c7: "arg3"
(gdb) x/s *(void**)(esp + 16)
0xffffd2c5: "2"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 2.20: -

### 3 Задания для самостоятельной работы

Программа из лабораторной 9, но с использованием подпрограмм (рис. 3.1)

```
%include 'in_out.asm'

SECTION .data
f_x db "функция: 10(x - 1)",0h
msg db 10,13,'результат: ',0h

SECTION .text
global _start

_f:
push ebx
dec eax
mov ebx, 10
mul ebx
pop ebx
ret

_start:
pop ecx
pop edx
sub ecx, 1
mov esi, 0

next:
cmp ecx,0h
jz _end
pop eax
self10.asm [+]
```

17

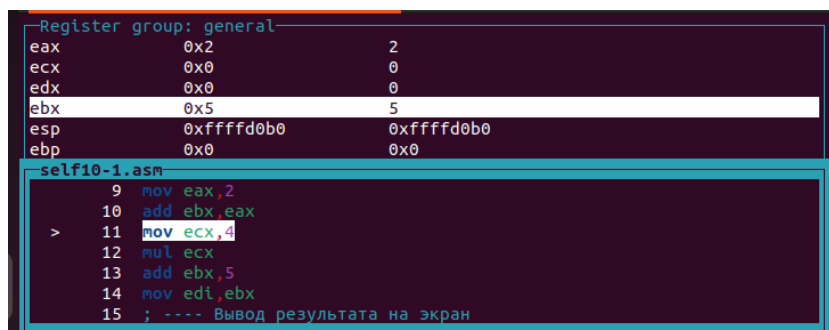
Рис. 3.1: -

и проверка ее работоспособности(рис. 3.2)

```
ssborunov@ssborunov-VirtualBox: ~/work/study/2022-2023/Arch-pc/study_2022-2023_a
rh-pc/labs/lab10$ ./self10
функция: 10(x - 1)
результат: 0
ssborunov@ssborunov-VirtualBox: ~/work/study/2022-2023/Arch-pc/study_2022-2023_a
rh-pc/labs/lab10$ ./self10 1 2 3 4
функция: 10(x - 1)
результат: 60
ssborunov@ssborunov-VirtualBox: ~/work/study/2022-2023/Arch-pc/study_2022-2023_a
rh-pc/labs/lab10$
```

Рис. 3.2: -

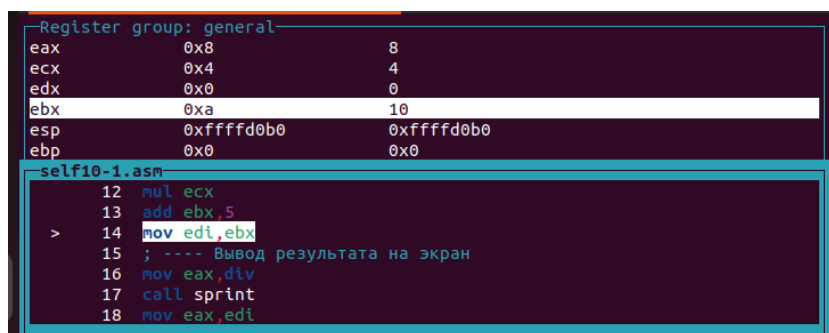
Просмотр регистров, для поиска ошибки в программе из листинга 10.3 (рис. 3.3) и (рис. 3.4)



```
Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x5      5
esp      0xffffd0b0 0xffffd0b0
ebp      0x0      0x0

self10-1.asm
9  mov eax, 2
10 add ebx, eax
11 mov ecx, 4
12 mul ecx
13 add ebx, 5
14 mov edi, ebx
15 ; ---- Вывод результата на экран
```

Рис. 3.3: -



```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa     10
esp      0xffffd0b0 0xffffd0b0
ebp      0x0      0x0

self10-1.asm
12 mul ecx
13 add ebx, 5
14 mov edi, ebx
15 ; ---- Вывод результата на экран
16 mov eax, div
17 call sprint
18 mov eax, edi
```

Рис. 3.4: -

Ошибка была в строчках

```
add ebx, eax
mov ecx, 4
mul ecx
add ebx, 5
mov edi, ebx
```

правильно работающая программа представлена на (рис. 3.5)



```

#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
~

```

Рис. 3.5: -

Проверка корректности работы программы, после исправлений (рис. 3.6)

```

(gdb) run
Starting program: /home/ssborunov/work/study/2022-2023/Arch-pc/study_2022-2023_
arch-pc/labs/lab10/self10-1
Результат: 25
[Inferior 1 (process 4710) exited normally]
(gdb) █

```

Рис. 3.6: -

## 4 Выводы

В результате выполнения работы, я научился организовывать код в подпрограммы и познакомился с базовыми функциями отладчика gdb.