

Отчёт по лабораторной работе №9

дисциплина: Архитектура компьютера

Гайдук Софья Сергеевна

Содержание

1 Цель работы	6
2 Выполнение лабораторной работы	7
3 Задания для самостоятельной работы	26
4 Выводы	36

Список иллюстраций

2.1	image1	7
2.2	image2	8
2.3	image3	8
2.4	image4	9
2.5	image5	9
2.6	image6	9
2.7	image7	10
2.8	image8	10
2.9	image9	11
2.10	image10	11
2.11	image11	12
2.12	image12	12
2.13	image13	13
2.14	image14	14
2.15	image15	14
2.16	image16	15
2.17	image17	15
2.18	image18	16
2.19	image19	17
2.20	image20	17
2.21	image21	18
2.22	image22	18
2.23	image23	19
2.24	image24	19
2.25	image25	20
2.26	image26	20
2.27	image27	21
2.28	image28	21
2.29	image29	21
2.30	image30	22
2.31	image31	22
2.32	image32	23
2.33	image33	23
2.34	image34	23
2.35	image35	24
2.36	image36	24

2.37	image37	24
2.38	image38	24
2.39	image39	25
2.40	image40	25
2.41	image41	25
3.1	image42	26
3.2	image43	26
3.3	image44	27
3.4	image45	27
3.5	image46	28
3.6	image47	28
3.7	image48	28
3.8	image49	29
3.9	image50	30
3.10	image51	31
3.11	image52	32
3.12	image53	33
3.13	image54	34
3.14	image55	34
3.15	image56	34
3.16	image57	35
3.17	image58	35

Список таблиц

1 Цель работы

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

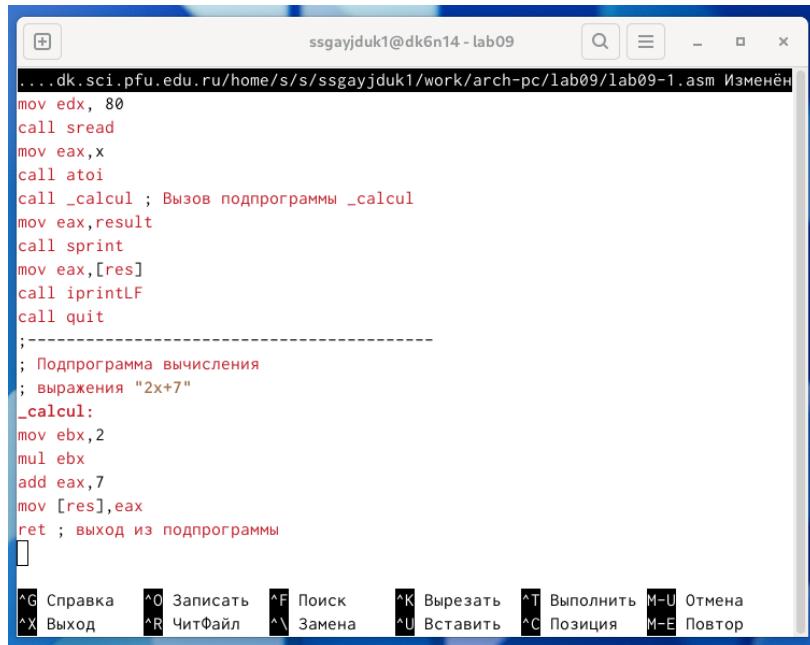
2 Выполнение лабораторной работы

Создадим каталог для выполнения лабораторной работы №9, перейдем в него, создадим файл lab09-1.asm и проверим его наличие (рис. 2.1).

```
ssgayjduk1@dk6n14 ~ $ mkdir ~/work/arch-pc/lab09
ssgayjduk1@dk6n14 ~ $ cd ~/work/arch-pc/lab09
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ touch lab09-1.asm
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ls
lab09-1.asm
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ 
```

Рисунок 2.1: image1

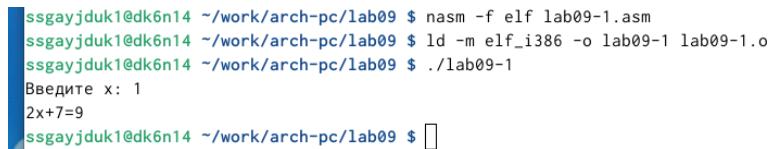
Введем в файл lab09-1.asm текст программы из листинга 9.1. Создадим исполняемый файл и проверим его работу (рис. 2.2).



```
ssgayjduk1@dk6n14 - lab09
....dk.sci.pfu.edu.ru/home/s/s/ssgayjduk1/work/arch-pc/lab09/lab09-1.asm Изменён
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax,result
call sprint
mov eax,[res]
call iprintLF
call quit
-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы
```

^G Справка ^O Записать ^F Поиск ^K Вырезать ^T Выполнить M-U Отмена
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^C Позиция M-E Повтор

Рисунок 2.2: image2



```
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 1
2x+7=9
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $
```

Рисунок 2.3: image3

Изменим текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где $f(x)=2x+7$ (выполняется в `_calcul`), $g(x)=3x-1$ (выполняется в `_subcalcul`). Создадим исполняемый файл и проверим его работу (рис. 2.4).

```
;-----  
; Подпрограмма вычисления  
; выражения "2x+7"  
_calcul:  
  
call _subcalcul  
mov eax,[res]  
mov ebx,2  
mul ebx  
add eax,7  
mov [res],eax  
ret ; выход из подпрограммы  
  
_subcalcul:  
mov ebx,3  
mul ebx  
sub eax,1  
mov [res],eax  
ret  
[]  
^G Справка ^O Записать ^F Поиск ^K Вырезать  
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить
```

Рисунок 2.4: image4

```
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm  
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o  
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ./lab09-1  
Введите x: 1  
2(3x-1)+7=11  
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ./lab09-1  
Введите x: 2  
2(3x-1)+7=17  
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ []
```

Рисунок 2.5: image5

Создайте файл lab09-2.asm с текстом программы из Листинга 9.2 (рис. 2.6).

```
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ touch lab09-2.asm  
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ls  
in_out.asm lab09-1 lab09-1.asm lab09-1.o lab09-2.asm  
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ []
```

Рисунок 2.6: image6

```
ssgayjduk1@dk6: ~
```

```
....dk.sci.pfu.edu.ru/home/s/s/ssgayjduk1/w
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
```

Рисунок 2.7: image7

Создадим исполняемый файл и проверим его работу (рис. 2.8).

```
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ nasm -f elf lab09-2.asm
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ./lab09-2
Hello, world!
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $
```

Рисунок 2.8: image8

Получим исполняемый файл для работы с GDB, в исполняемый файл необходимо добавить отладочную информацию. Трансляцию программ необходимо прово-

дить с ключом „-g“. Загрузим исполняемый файл в отладчик gdb (рис. 2.9).

```
ssgayjduk1@dk6n14 ~ $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
nasm: fatal: unable to open input file 'lab09-2.asm' No such file or directory
ssgayjduk1@dk6n14 ~ $ cd ~/work/arch-pc/lab09
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ gdb lab09-2
Exception caught while booting Guile.
Error in function "load-thunk-from-memory":
missing DT_GUILDE_ENTRY
gdb: warning: Could not complete Guile gdb module initialization from:
/usr/share/gdb/guile/gdb/boot.scm.
Limited Guile support is available.
Suggest passing --data-directory=/path/to/gdb/data-directory.
GNU gdb (Gentoo 16.3 vanilla) 16.3
Copyright (C) 2024 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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 lab09-2...
(gdb) run
```

Рисунок 2.9: image9

Проверим работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 2.10).

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/s/ssgayjduk1/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 4451) exited normally]
(gdb) 
```

Рисунок 2.10: image10

Для более подробного анализа программы установим брейкпойнт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустим её (рис. 2.11).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/s/sgayjduk1/work/arch-pc/lab09/lab
09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb) 
```

Рисунок 2.11: image11

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки _start (рис. 2.12).

```
(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) 
```

Рисунок 2.12: image12

Переключимся на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel (рис. 2.13).

```
(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) █
```

Рисунок 2.13: image13

Различия отображения синтаксиса машинных команд в режимах ATT и Intel заключаются в обозначении регистров % и значений \$, которые в GDB используются по умолчанию.

Включим режим псевдографики для более удобного анализа программы (рис. 2.14).

```
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
0x8049031 <_start+49>   mov    ebx,0x0
0x8049036 <_start+54>   int    0x80
0x8049038           add    BYTE PTR [eax],al

native process 4531 (asm) In: _start
(gdb) █
```

Рисунок 2.14: image14

```
Register group: general
eax      0x0          0
ecx      0x0          0
edx      0x0          0
ebx      0x0          0
esp     0xfffffc360  0xfffffc360
ebp      0x0          0x0
esi      0x0          0
edi      0x0          0
eip     0x8049000  0x8049000 <_start>

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 4531 (asm) In: _start
(gdb) layout regs
(gdb) █
```

Рисунок 2.15: image15

На предыдущих шагах была установлена точка останова по имени метки (`_start`).

Проверим это с помощью команды `info breakpoints (i b)` (рис. 2.16).

```
(gdb) i b
Num      Type            Disp Enb Address      What
1       breakpoint      keep y  0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) 
```

Рисунок 2.16: image16

```
ssgayjduk1@dk6n14 - lab09
Register group: general
eax          0x0          0
ecx          0x0          0
edx          0x0          0
ebx          0x0          0
esp 0xfffffc360 0xfffffc360
ebp          0x0          0x0
esi          0x0          0
edi          0x0          0
eip 0x8049000 0x8049000 <_start>
eflags        0x202      [ IF ]
cs           0x23        35
ss           0x2b        43

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
b+ 0x8049031 <_start+49> mov  ebx,0x0

native process 5034 (asm) In: _start          L9    PC: 0x8049000
(gdb) layout regs
(gdb) i b
Num      Type            Disp Enb Address      What
1       breakpoint      keep y  0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) 
```

Рисунок 2.17: image17

Определим адрес предпоследней инструкции (mov ebx,0x0) и установите точку останова. Посмотрите информацию о всех установленных точках останова с помощью i b (рис. 2.18).

The screenshot shows a GDB session window titled "ssgayjduk1@dk6n14 - lab09".

Registers:

Register	Value	Description
eax	0x0	
ecx	0x0	
edx	0x0	
ebx	0x0	
esp	0xfffffc360	0xfffffc360
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049000	0x8049000 <_start>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43

Assembly:

```
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
b+ 0x8049031 <_start+49>  mov    ebx,0x0
```

Breakpoints:

```
native process 5034 (asm) In: _start          L9    PC: 0x8049000
(gdb) layout regs
(gdb) i b
Num      Type            Disp Enb Address      What
1        breakpoint      keep y  0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num      Type            Disp Enb Address      What
1        breakpoint      keep y  0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
2        breakpoint      keep y  0x08049031 lab09-2.asm:20
(gdb) 
```

Рисунок 2.18: image18

Выполним 5 инструкций с помощью команды stepi (si) и проследим за изменением значений регистров (рис. 2.19).

The screenshot shows a debugger interface with the following details:

- Registers:** General registers (eax, ecx, edx, ebx, esp, ebp, esi, edi) are listed with their current values: eax=0x4, ecx=0x0, edx=0x0, ebx=0x0, esp=0xfffffc360, ebp=0x0, esi=0x0, edi=0x0.
- Assembly Code:** The assembly code is shown in a scrollable window:

```
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
```
- Native Process:** native process 3528 (asm) In: _start
- Registers at PC:** L10 PC: 0x8049005
- GDB Commands:** (gdb) si

Рисунок 2.19: image19

The screenshot shows a debugger interface with the following details:

- Registers:** General registers (eax, ecx, edx, ebx, esp, ebp, esi, edi) are listed with their current values: eax=0x4, ecx=0x0, edx=0x0, ebx=0x1, esp=0xfffffc360, ebp=0x0, esi=0x0, edi=0x0.
- Assembly Code:** The assembly code is shown in a scrollable window:

```
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
```
- Native Process:** native process 3528 (asm) In: _start
- Registers at PC:** L11 PC: 0x804900a
- GDB Commands:** (gdb) si

Рисунок 2.20: image20

The screenshot shows a debugger interface with the following details:

- Registers:** General registers (eax, ecx, edx, ebx, esp, ebp, esi, edi) are listed with their current values.
- Assembly:** The assembly code starts at address 0x8049000 and includes instructions like mov eax, 0x4; mov ebx, 0x1; mov ecx, 0x804a000; etc.
- Stack:** The stack pointer (esp) is at 0xfffffc360.
- Call Stack:** The call stack shows native process 3528 (asm) In: _start.
- Registers:** The edx register is highlighted in red.
- Registers:** The edx register is highlighted in red.

Рисунок 2.21: image21

The screenshot shows a debugger interface with the following details:

- Registers:** General registers (eax, ecx, edx, ebx, esp, ebp, esi, edi) are listed with their current values.
- Assembly:** The assembly code starts at address 0x8049000 and includes instructions like mov eax, 0x4; mov ebx, 0x1; mov ecx, 0x804a000; etc.
- Stack:** The stack pointer (esp) is at 0xfffffc360.
- Call Stack:** The call stack shows native process 3528 (asm) In: _start.
- Registers:** The edx register is highlighted in red.
- Registers:** The edx register is highlighted in red.

Рисунок 2.22: image22

The screenshot shows a debugger interface with the following details:

- Registers:**

	eax	ecx	edx	ebx	esp	ebp	esi	edi
eax	0x8	0x804a000	134520832					
ecx	0x8		8					
edx	0x1			1				
ebx	0x0				0xfffffc360			
esp	0x0					0x0		
ebp	0x0						0x0	
esi	0x0						0	
edi	0x0							0
- Assembly Code:**

```

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

```
- Command Line:**

```

native process 3528 (asm) In: _start
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) 

```
- Status Bar:**

L14 PC: 0x8049016

Рисунок 2.23: image23

Изменяются значения регистров ebx, ecx, edx, на eax мы заканчиваем выполнение инструкций.

Посмотреть содержимое регистров также можно с помощью команды info registers (i r) (рис. 2.24).

The screenshot shows a debugger interface with the following details:

- Registers:**

	eax	ecx	edx	ebx	esp	ebp	esi	edi
eax	0x8	0x804a000	134520832					
ecx	0x8		8					
edx	0x1			1				
ebx	0x0				0xfffffc360			
esp	0x0					0x0		
ebp	0x0						0x0	
esi	0x0						0	
edi	0x0							0
- Assembly Code:**

```

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

```
- Command Line:**

```

native process 3528 (asm) In: _start
(gdb) i r

```
- Status Bar:**

L14 PC: 0x8049016

Рисунок 2.24: image24

The screenshot shows a debugger interface with the following details:

- Registers:**

	eax	ecx	edx	ebx	esp	ebp	esi	edi
eax	0x8	0x804a000	134520832					
ecx	0x8		8					
edx	0x1			1				
ebx	0x0				0xfffffc360			
esp	0x0					0xfffffc360		
ebp	0x0					0x0		
esi	0x0						0	
edi	0x0							0
- Assembly Code:**

```

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

```
- Registers (native process):**

	eax	ecx	edx	ebx	esp	ebp	esi	edi	eip
eax	0x8	0x804a000	134520832						L14
ecx	0x8		8						PC: 0x8049016
edx	0x1			1					
ebx	0x0				0xfffffc360				
esp	0x0					0xfffffc360			
ebp	0x0					0x0			
esi	0x0						0		
edi	0x0							0	
eip	0x8049016								0x8049016 <_start+22>
- Command Line:**

```
--Type <RET> for more, q to quit, c to continue without paging--
```

Рисунок 2.25: image25

Посмотрим значение переменной msg1 по имени (рис. 2.26).

The screenshot shows a debugger interface with the following details:

- Registers:**

	eax	ecx	edx	ebx	esp	ebp	esi	edi
eax	0x8	0x804a000	134520832					
ecx	0x8		8					
edx	0x1			1				
ebx	0x0				0xfffffc360			
esp	0x0					0xfffffc360		
ebp	0x0					0x0		
esi	0x0						0	
edi	0x0							0
- Assembly Code:**

```

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

```
- Registers (native process):**

	eax	ecx	edx	ebx	esp	ebp	esi	edi	eip
eax	0x8	0x804a000	134520832						L14
ecx	0x8		8						PC: 0x8049016
edx	0x1			1					
ebx	0x0				0xfffffc360				
esp	0x0					0xfffffc360			
ebp	0x0					0x0			
esi	0x0						0		
edi	0x0							0	
- Command Line:**

```
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) 
```

Рисунок 2.26: image26

Посмотрим значение переменной msg2 по адресу. Посмотрим инструкцию тов ecx,msg2 которая записывает в регистр ecx адрес перемененной msg2 (0x804a008)

(рис. 2.27).

The screenshot shows a debugger interface with the title "ssgayiduk1@dk6n14 - lab09". The "Registers" tab is selected, displaying the following register values:

eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xfffffc360	0xfffffc360
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 <_start+22>
eflags	0x202	[IF]

The assembly code window shows the following instructions:

```
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
```

The command line shows:

```
native process 3528 (asm) In: _start
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) 
```

Рисунок 2.27: image27

Изменим первый символ переменной msg1 (рис. 2.28).

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

Рисунок 2.28: image28

Изменим первый символ переменной msg2 (рис. 2.29).

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

Рисунок 2.29: image29

Изменим второй символ переменной msg1 и первый и четвертый символ переменной msg2 (рис. 2.30).

```
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hhllo, "
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "World!\n\034"
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "Lor d!\n\034"
(gdb) 
```

Рисунок 2.30: image30

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. 2.31).

```
(gdb) p/x $edx
$6 = 0x8
(gdb) p/t $edx
$7 = 1000
(gdb) p/s $edx
$8 = 8
(gdb) 
```

Рисунок 2.31: image31

С помощью команды set изменим значение регистра ebx (рис. 2.32).

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

Рисунок 2.32: image32

```
(gdb) set $ebx=2  
(gdb) p/s $ebx  
$5 = 2  
(gdb) █
```

Рисунок 2.33: image33

Разница вывода команд p/s \$ebx в том, что „2“ мы передаем строку, код которой преобразуем в символьной вид, а 2 мы передаем число, которое также преобразуем в символьной вид.

Завершим выполнение программы с помощью команды continue (c) (рис. 2.34).

```
(gdb) c  
Continuing.  
Lor d!  
  
Breakpoint 2, _start () at lab09-2.asm:20  
(gdb) █
```

Рисунок 2.34: image34

И выйдем из GDB с помощью команды quit (q) (рис. 2.35).

```
(gdb) q
A debugging session is active.

      Inferior 1 [process 4371] will be killed.

Quit anyway? (y or n) y
```

Рисунок 2.35: image35

Скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm (рис. 2.36).

```
ssgayjduk1@dk6n14 ~ $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
```

Рисунок 2.36: image36

Создаем исполняемый файл (рис. 2.37).

```
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $
```

Рисунок 2.37: image37

Загрузим исполняемый файл в отладчик, указав аргументы (рис. 2.38).

```
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
Exception caught while booting Guile.
Error in function "load-thunk-from-memory":
missing DT_GUILLE_ENTRY
gdb: warning: Could not complete Guile gdb module initialization from:
/usr/share/edk/guile/edk/boot scm
```

Рисунок 2.38: image38

Исследуем расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установим точку останова перед первой инструкцией в программе и запустим ее (рис. 2.39).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 8.
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/ssgayjduk1/work/arch-pc/lab09/lab09-3 аргумент 1 аргумент 2 аргумент\ 3
Breakpoint 1, _start () at lab09-3.asm:8
8      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) 
```

Рисунок 2.39: image39

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы) (рис. 2.40).

```
(gdb) x/x $esp
0xfffffc310: 0x00000005
(gdb) 
```

Рисунок 2.40: image40

Число аргументов равно 5 - это имя и четыре аргумента.

Посмотрите остальные позиции стека (рис. 2.41).

```
(gdb) x/s *(void**)( $esp + 4)
0xfffffc57d: "/afs/.dk.sci.pfu.edu.ru/home/s/ssgayjduk1/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xfffffc5c4: "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xfffffc5d6: "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xfffffc5e7: "2"
(gdb) x/s *(void**)( $esp + 20)
0xfffffc5e9: "аргумент 3"
(gdb) x/s *(void**)( $esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb) 
```

Рисунок 2.41: image41

Шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.) потому, что в esp находится 32-битной системе, где размер одного аргумента равен 4 байта.

3 Задания для самостоятельной работы

Номер 1.

Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. Скопируем файл из лабораторной работы №8 с названием lab09-4.asm. Создадим исполняемый файл и проверим его работу (рис. 3.1).

```
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab09/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ nasm -f elf lab09-4.asm
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-4 lab09-4.o
```

Рисунок 3.1: image42

```
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ./lab09-4 1 2 3 4
Функция: f(x)=4x-3
Результат: 28
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ./lab09-4 1 1 1 1
Функция: f(x)=4x-3
Результат: 4
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ./lab09-4 4 5 6 7
Функция: f(x)=4x-3
Результат: 76
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $
```

Рисунок 3.2: image43

The screenshot shows a terminal window with the following content:

```
mc [ssgayjduk1@dk6n14.dk.sci.pfu.edu.ru]:~/work/arch-pc/lab09
/afs/.dk.sci.pfu.edu.ru/home/s/s/ssgayjduk1/work/arch-pc/lab09/lab09-4.asm
%include 'in_out.asm'
SECTION .data
msg1 db "Функция: f(x)=4x-3",0
msg2 db "Результат: ",0
msg3 db "Номер варианта 6",0
SECTION .text
global _start
_start:
    pop ecx ; Извлекаем из стека в `ecx` количество аргументов (первое значение в с)
    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,1 ; Используем `esi` для хранения промежуточных значений функций
    mov edi,0 ; Сумма функций
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

loop next ; переход к обработке следующего аргумента
_end:
    mov eax, msg3 ; вывод сообщения "Номер варианта 6"
    mov eax, msg1 ; вывод сообщения "Функция: f(x)=4x-3 "
    call sprintLF
    mov eax, msg2 ; вывод сообщения "Результат: "
    call sprint
    mov eax, edi ; записываем сумму в регистр `eax`
    call iprintLF ; печать результата
    call quit ; завершение программы

_calcul:
    mov eax, eax
    mov ebx, 4
    mul ebx
    sub eax, 3
    add edi, eax

ret
```

At the bottom of the terminal window, there is a menu bar with Russian labels: Справка, Записать, Поиск, Вырезать, Выполнить, Отмена.

Рисунок 3.3: image44

Номер 2.

Создадим файл с названием lab09-5.asm. Проверьте программу. С помощью отладчика GDB, анализируя изменения значений регистров (рис. 3.4).

```
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ touch lab09-5.asm
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $
```

Рисунок 3.4: image45

```
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-5.lst lab09-5.asm
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ gdb lab09-5
```

Рисунок 3.5: image46

```
Reading symbols from lab09-5...
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/s/ssgayjduk1/work/arch-pc/lab09/lab09-5
Результат: 10
[Inferior 1 (process 4386) exited normally]
(gdb) 
```

Рисунок 3.6: image47

The screenshot shows the GDB interface with the following details:

- Registers:** A table showing general registers:

Register	Value	Description
eax	0x2	2
ecx	0x0	0
edx	0x0	0
ebx	0x5	5
esp	0xfffffc320	0xfffffc320
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80490f4	0x80490f4 <_start+12>
- Assembly:** The assembly code for the program:

```
B+ 0x80490e8 <_start>    mov    ebx,0x3
 0x80490ed <_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
 0x80490fe <_start+22>   mov    edi,ebx
 0x8049100 <_start+24>   mov    eax,0x804a000
 0x8049105 <_start+29>   call   0x804900f <sprint>
 0x804910a <_start+34>   mov    eax,edi
```
- Registers:** A command history showing register operations:

```
native process 5068 (asm) In: _start
(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) 
```

Рисунок 3.7: image48

The screenshot shows a debugger interface with the following details:

- Registers:** A table showing general registers:

	eax	ecx	edx	ebx	esp	ebp	esi	edi	eip
Value	0x8	0x4	0x0	0x5	0xfffffc320	0x0	0x0	0x0	0x80490fb
Description	8	4	0	5	0xfffffc320	0x0	0	0	<_start+19>
- Assembly Code:** The assembly code is displayed in a scrollable window:

```
B+ 0x80490e8 <_start>    mov    ebx,0x3
0x80490ed <_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
0x80490fe <_start+22>   mov    edi,ebx
0x8049100 <_start+24>   mov    eax,0x804a000
0x8049105 <_start+29>   call   0x804900f <sprint>
0x804910a <_start+34>   mov    eax,edi
```
- Native process:** Information about the native process:

native process 5068 (asm) In: _start
- Commands:** A history of GDB commands:

(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) []
- Registers:** Shows the current PC value:

L13 PC: 0x80490fb

Рисунок 3.8: image49

The screenshot shows a debugger window with the following details:

- Registers:**

	Register	Value
eax	0x8	8
ecx	0x4	4
edx	0x0	0
ebx	0xa	10
esp	0xfffffc320	0xfffffc320
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80490fe	0x80490fe <_start+22>
- Assembly Code:**

```

B+ 0x80490e8 <_start>    mov    ebx,0x3
0x80490ed <_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
>0x80490fe <_start+22>  mov    edi,ebx
0x8049100 <_start+24>   mov    eax,0x804a000
0x8049105 <_start+29>   call   0x804900f <sprint>
0x804910a <_start+34>   mov    eax,edi

```
- Command Line:**

```

native process 5068 (asm) In: _start
(gdb) layout regs
(gdb) si
(gdb) []

```
- Status Bar:**

L14 PC: 0x80490fe

Рисунок 3.9: image50

Определим ошибку и исправим ее. Мы видим, что значения изначально записываются в регистр ebx, а умножение происходит в регистре eax. Далее ebx прибавляем 5 и выводится результат ebx. Исправим это. После суммы eax и ebx перезапишем ее в eax. Прибавляем 5 в eax, и вывод в eax (рис. 3.10).

The screenshot shows a debugger interface with the following details:

- Registers:** A table showing general registers:

Register	Value	Description
eax	0x2	2
ecx	0x0	0
edx	0x0	0
ebx	0x5	5
esp	0xfffffc320	0xfffffc320
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80490f4	0x80490f4 <_start+12>
- Assembly Code:** The assembly code being executed:

```
B+ 0x80490e8 <_start>    mov    ebx,0x3
0x80490ed <_start+5>    mov    eax,0x2
0x80490f2 <_start+10>   add    ebx,eax
>0x80490f4 <_start+12>  mov    eax,ebx
0x80490f6 <_start+14>   mov    ecx,0x4
0x80490fb <_start+19>   mul    ecx
0x80490fd <_start+21>   add    eax,0x5
0x8049100 <_start+24>   mov    edi,eax
0x8049102 <_start+26>   mov    eax,0x804a000
0x8049107 <_start+31>   call   0x804900f <sprint>
```
- Registers View:** A command-line interface for viewing register values:

```
native process 5615 (asm) In: _start
(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) []
```

Рисунок 3.10: image51

The screenshot shows a debugger window with the following details:

- Registers:** A table showing general registers:

	eax	ecx	edx	ebx	esp	ebp	esi	edi	eip
0x5	0x0	0x0	0x0	0x5	0xfffffc320	0x0	0x0	0x0	0x80490f6
5	0	0	0	5	0xfffffc320	0x0	0	0	0x80490f6 <_start+14>
- Assembly:** A list of assembly instructions:

```
B+ 0x80490e8 <_start>    mov    ebx,0x3
0x80490ed <_start+5>    mov    eax,0x2
0x80490f2 <_start+10>   add    ebx,eax
0x80490f4 <_start+12>   mov    eax,ebx
>0x80490f6 <_start+14>  mov    ecx,0x4
0x80490fb <_start+19>   mul    ecx
0x80490fd <_start+21>   add    eax,0x5
0x8049100 <_start+24>   mov    edi,eax
0x8049102 <_start+26>   mov    eax,0x804a000
0x8049107 <_start+31>   call   0x804900f <sprint>
```
- Command Line:** Shows native process information and GDB commands:

```
native process 5615 (asm) In: _start
(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```
- Status:** L13 PC: 0x80490f6

Рисунок 3.11: image52

The screenshot shows a debugger window with the following details:

- Registers:** A table showing general registers:

	eax	0x14	20
ecx	0x4	4	
edx	0x0	0	
ebx	0x5	5	
esp	0xfffffc320	0xfffffc320	
ebp	0x0	0x0	
esi	0x0	0	
edi	0x0	0	
eip	0x80490fd	0x80490fd	<_start+21>
- Assembly:** The assembly code for the program:

```
B+ 0x80490e8 <_start>    mov    ebx,0x3
0x80490ed <_start+5>    mov    eax,0x2
0x80490f2 <_start+10>   add    ebx,eax
0x80490f4 <_start+12>   mov    eax,ebx
0x80490f6 <_start+14>   mov    ecx,0x4
0x80490fb <_start+19>  mul    ecx
>0x80490fd <_start+21> add    eax,0x5
0x8049100 <_start+24>  mov    edi,eax
0x8049102 <_start+26>  mov    eax,0x804a000
0x8049107 <_start+31>  call   0x804900f <sprint>
```
- Native process:** Information about the native process:

```
native process 5615 (asm) In: _start
```
- GDB Session:** A series of GDB commands:

```
(gdb) layout regs
(gdb) si
(gdb)
```
- Registers:** Shows the current PC value:

```
L15  PC: 0x80490fd
```

Рисунок 3.12: image53

The screenshot shows a debugger window with the title "ssgayjduk1@dk6n14 - lab09". It displays the assembly code for the program, starting from the address 0x80490e8. The assembly code includes instructions like mov, add, and call. Below the assembly code, there is a register dump titled "Register group: general" showing values for eax, ecx, edx, ebx, esp, ebp, esi, edi, and eip. The eip register is highlighted with a blue bar and contains the value 0x8049100, which corresponds to the label <_start+24>. The bottom part of the window shows a command-line interface with the prompt "(gdb)" and several "si" commands followed by a "c" command to continue execution.

```

Register group: general
eax          0x19          25
ecx          0x4           4
edx          0x0           0
ebx          0x5           5
esp          0xfffffc320    0xfffffc320
ebp          0x0           0x0
esi          0x0           0
edi          0x0           0
eip          0x8049100      0x8049100 <_start+24>

B+ 0x80490e8 <_start>    mov   ebx,0x3
0x80490ed <_start+5>    mov   eax,0x2
0x80490f2 <_start+10>   add   ebx,eax
0x80490f4 <_start+12>   mov   eax,ebx
0x80490f6 <_start+14>   mov   ecx,0x4
0x80490fb <_start+19>   mul   ecx
0x80490fd <_start+21>   add   eax,0x5
>0x8049100 <_start+24>  mov   edi,eax
0x8049102 <_start+26>   mov   eax,0x804a000
0x8049107 <_start+31>   call  0x804900f <sprint>

native process 5615 (asm) In: _start
(gdb) layout regs
(gdb) si
(gdb) c
(gdb) []

```

Рисунок 3.13: image54

```

(gdb) layout regs
(gdb) si
(gdb) c
Continuing.
Результат: 25
(gdb) [бр 1 (process 5615) exited normally]

```

Рисунок 3.14: image55

Теперь результат правильный.

Создадим исполняемый файл и проверим его работу (рис. 3.15).

```

ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-5.lst lab09-5.asm
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
ssgayjduk1@dk6n14 ~/work/arch-pc/lab09 $ gdb lab09-5

```

Рисунок 3.15: image56

```
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/s/ssgayjduk1/work/arch-pc/lab09/lab09-5
Результат: 25
[Inferior 1 (process 6058) exited normally]
(gdb) 
```

Рисунок 3.16: image57

Отправим файлы на Github (рис. 3.17).

```
09/report $ git add .
ssgayjduk1@dk6n14 ~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab
09/report $ git commit -am 'feat(main): add lab-9'
[master f4bdb1b] feat(main): add lab-9
 3 files changed, 128 deletions(-)
 delete mode 100644 labs/lab09/report/_quarto.yml
 delete mode 100644 labs/lab09/report/arch-pc--lab09--report.qmd
 delete mode 100644 labs/lab09/report/image/solvay.jpg
ssgayjduk1@dk6n14 ~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab
09/report $ git push
Перечисление объектов: 9, готово.
Подсчет объектов: 100% (9/9), готово.
При сжатии изменений используется до 6 потоков
Сжатие объектов: 100% (5/5), готово.
Запись объектов: 100% (5/5), 449 байтов | 449.00 КиБ/с, готово.
Total 5 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To github.com:SofiaGayduk/study_2025-2026_arh-pc.git
  e9df6b9..f4bdb1b master -> master
ssgayjduk1@dk6n14 ~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab
09/report $ 
```

Рисунок 3.17: image58

4 Выводы

Мы приобрели навыки написания программ с использованием подпрограмм. Ознакомились с методами отладки при помощи GDB и его основными возможностями.