

# **Лабораторная работа №9**

**Понятие подпрограммы. Отладчик GDB**

Глобин Никита Анатольевич

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
3.1	Выполнение задания для самостоятельной работы . . . . .	15
<b>4</b>	<b>Выводы</b>	<b>20</b>
	<b>Список литературы</b>	<b>21</b>

## Список иллюстраций

3.1 фото 1 . . . . .	7
3.2 фото 2 . . . . .	7
3.3 фото 3 . . . . .	8
3.4 фото 4 . . . . .	8
3.5 фото 5 . . . . .	9
3.6 фото 6 . . . . .	10
3.7 фото 7 . . . . .	10
3.8 фото 8 . . . . .	10
3.9 фото 9 . . . . .	11
3.10 фото 10 . . . . .	11
3.11 фото 11 . . . . .	12
3.12 фото 12 . . . . .	12
3.13 фото 13 . . . . .	12
3.14 фото 14 . . . . .	12
3.15 фото 15 . . . . .	13
3.16 фото 16 . . . . .	13
3.17 фото 18 . . . . .	13
3.18 фото 20 . . . . .	14
3.19 фото 21 . . . . .	14
3.20 фото 22 . . . . .	15
3.21 фото 23 . . . . .	15
3.22 фото 24 . . . . .	16
3.23 фото 25 . . . . .	16
3.24 фото 26 . . . . .	16
3.25 фото 27 . . . . .	17
3.26 фото 28 . . . . .	17
3.27 фото 29 . . . . .	18
3.28 фото 30 . . . . .	18
3.29 фото 31 . . . . .	19
3.30 фото 32 . . . . .	19

## **Список таблиц**

# 1 Цель работы

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

## **2 Задание**

1. Цель работы
2. Выполнение лабораторной работы
3. Выполнение задания для самостоятельной работы
4. Выводы

## 3 Выполнение лабораторной работы

1. создаём файл и переписываем в него код (рис. 3.1).

```
%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:
;-----
; 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, [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
```

Рис. 3.1: фото 1

компилируем и запускаем (рис. 3.2).

```
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$ touch lab09-1.asm
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$ nasm -f elf lab09-1.asm
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$ ld -m elf_i386 -o lab09 lab09-1.o
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$ ./lab09
Введите x: 3
2x+7=13
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$
```

Рис. 3.2: фото 2

изменяем программу (рис. 3.3).

```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(g(x))=',0
SECTION .bss
x: RESB 80
res: 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, [res]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret
```

Рис. 3.3: фото 3

компилируем и запускаем (рис. 3.4).

```
dodogvbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$ nasm -f elf lab09-1.asm
dodogvbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$ ld -m elf_i386 -o lab09 lab09-1.o
dodogvbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$ ./lab09
Введите x: 3
f(g(x))=23
dodogvbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$
```

Рис. 3.4: фото 4



2. создаём файл и переписываем в него код (рис. 3.5).

```
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
```

Рис. 3.5: фото 5

запускаем отладчик(рис. 3.6).

```

dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$ touch lab09-2.asm
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 15.2-3.fc41
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-redhat-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 lab09-2...
(gdb) run
Starting program: /home/dodo/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading 47.72 K separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 6372) exited normally]
(gdb) run
Starting program: /home/dodo/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09/lab09-2
Hello, world!
[Inferior 1 (process 6452) exited normally]
(gdb)

```

Рис. 3.6: фото 6

Проверьте работу программы, запустив ее в оболочке GDB (рис. 3.7).

```

Starting program: /home/dodo/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09/lab09-2
Hello, world!
[Inferior 1 (process 6452) exited normally]
(gdb) break _start
Breakpoint 1 at 0x08049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/dodo/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09/lab09-2

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

```

Рис. 3.7: фото 7

Посмотрим дисассимилированный код программы (рис. 3.8).

```

(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. 3.8: фото 8

переключим на отображение команд с Intel'овским синтаксисом (рис. 3.9).

```
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.
(gdb) █
```

Рис. 3.9: фото 9

### 3. Добавление точек останова (рис. 3.10).

```
Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffce40 0xffffce40  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

native process 6637 (asm) In: _start
(gdb) layout regs
(gdb)
```

Рис. 3.10: фото 10

Посмотрите значение переменной msg1 по имени (рис. 3.11) (рис. 3.12).

```
(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)
```

Рис. 3.11: фото 11

```
breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb)
```

Рис. 3.12: фото 12

узнаем значение ячейки (рис. 3.13).

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

Рис. 3.13: фото 13

поменяем значение ячейки (рис. 3.14).

```
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hhlllo, "
(gdb)
```

Рис. 3.14: фото 14

Выведете в различных форматах значение регистра edx (рис. 3.15).

```

0x804a000 <msg1>:      "Hhlllo, "
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) 

```

Рис. 3.15: фото 15

```

$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb) 

```

Рис. 3.16: фото 16

Разница в выводе p/s \$edx зависит от содержимого регистра \$edx:

Если значение регистра указывает на строку, выводится строка.

Если значение регистра — просто число или некорректный адрес, вывод будет ошибочным либо числовым.

## 5. Обработка аргументов командной строки в GDB

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  
компилируем файл (рис. 3.17).

```

dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab09$

```

Рис. 3.17: фото 18

запускаем отладчик для программы (рис. 3.18).

```
dodo@vbox: ~/work/study/2023-2024/Архитектура компьютера/arch-pc/arch--pc/labs/lab09
мент 2 'аргумент 3'
GNU gdb (Fedora Linux) 15.2-3.fc41
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-redhat-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 lab09-3...
(gdb) 
```

Рис. 3.18: фото 20

Для начала установим точку останова перед первой инструкцией в программе и запустим ее (рис. 3.19).

```
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-redhat-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 lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 7.
(gdb) run
Starting program: /home/dodo/work/study/2023-2024/Архитектура компьютера/arch-pc/arch--pc/labs/lab09/lab09-3 аргумент1
аргумент 2 аргумент\ 3

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

Breakpoint 1, _start () at lab09-3.asm:7
7      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xfffff000: 0x00000005
(gdb) 
```

Рис. 3.19: фото 21

проведем запуск программы при разных условиях (рис. 3.20).

```

Breakpoint 1, _start () at lab09-3.asm:7
7       pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffce00: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffcfe6: "/home/dodo/work/study/2023-2024/Архитектура компьютера/arch-pc/arch--pc/labs/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd055: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd067: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd078: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd07a: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
xxx: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 3.20: фото 22

Шаг изменения адреса равен 4 байта из-за: 1. 32-битной архитектуры, где указатели имеют размер 4 байта. 2. Организации стека, использующего выравнивание по границе слова для указателей.

### 3.1 Выполнение задания для самостоятельной работы

Скопируем файл первого задания прошлой самостоятельной работы. Нам нужно переписать его так, чтобы он использовал для авчисления выражения подпрограмму (рис. 3.21).

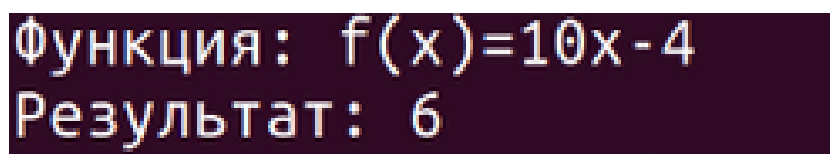
```

#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg2 db "Функция: f(x)=10x-4",0
SECTION .text
global _start
_start:
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

```

Рис. 3.21: фото 23

Соберём его и проверим корректность выполнения (рис. 3.22).



```
Функция:  $f(x)=10x-4$   
Результат: 6
```

Рис. 3.22: фото 24

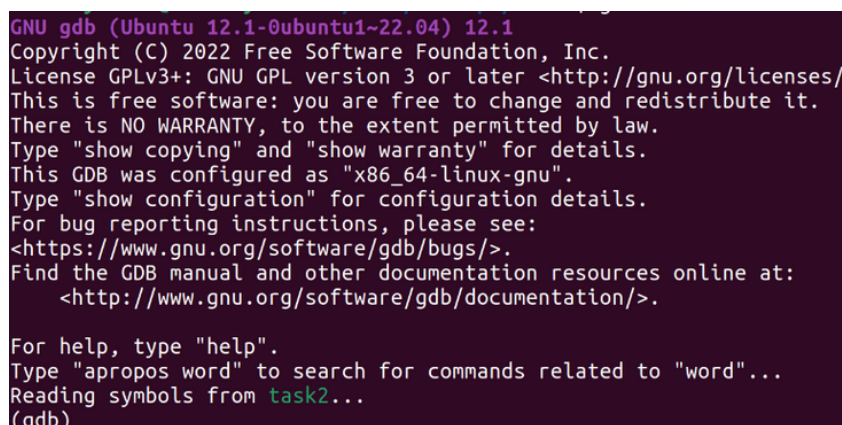
Создадим файл второго задания самостоятельной работы и вставим в него код из листинга 9.3. А затем соберём его и запустим (рис. 3.23).



```
Результат: 10
```

Рис. 3.23: фото 25

Как видим, код считает значение выражения неправильно. Загрузим его в gdb (рис. 3.24).



```
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1  
Copyright (C) 2022 Free Software Foundation, Inc.  
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/  
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 task2...  
(gdb)
```

Рис. 3.24: фото 26

Переключим его на синтаксис intel. Включим графическое отображение кода. Включением графическое отображение значений регистров. Установим брейкпоинт на `_start` и начнём построчно выполнять код



```

--Register group: general--
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd110 0xffffd110

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

native process 3644 In: _start L8 PC: 0>
(gdb) layout regs
(gdb) break _start
Breakpoint 1 at 0x80490e8: file task2.asm, line 8.
(gdb) run
Starting program: /home/nsandryushin/work/arch-pc/lab09/task2

Breakpoint 1, _start () at task2.asm:8
(gdb) █

```

Рис. 3.25: фото 27

```

--Register group: general--
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd110 0xffffd110

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

native process 3644 In: _start L9 PC: 0>
(gdb) break _start
Breakpoint 1 at 0x80490e8: file task2.asm, line 8.
(gdb) run
Starting program: /home/nsandryushin/work/arch-pc/lab09/task2

Breakpoint 1, _start () at task2.asm:8
(gdb) si

```

Рис. 3.26: фото 28

```

eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd110 0xffffd110

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

native process 3644 In: _start L10 PC: 0
Breakpoint 1 at 0x80490e8: file task2.asm, line 8.
(gdb) run
Starting program: /home/nsandryushin/work/arch-pc/lab09/task2

Breakpoint 1, _start () at task2.asm:8
(gdb) si
(gdb) si
(gdb) si

```

Рис. 3.27: фото 29

```

eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd110 0xffffd110

   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>

native process 3644 In: _start L14 PC: 0
Breakpoint 1, _start () at task2.asm:8
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si

```

Рис. 3.28: фото 30

Как видим, мы должны были умножить значение регистра ebx, но умножили регистр eax. Нам необходимо все результаты хранить в регистре eax. Изменим код (рис. 3.29).

```

%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. 3.29: фото 31

И проверим корректность его выполнения (рис. 3.30).



Результат: 25

Рис. 3.30: фото 32

## 4 Выводы

В результате выполнения лабораторной работы были получены представления о работе подпрограмм, а также было реализовано несколько программ, использующих подпрограммы. Также, были получены навыки работы с базовым функционалом gdb, и с помощью gdb была отловлена ошибка в коде программы. Я уверен что эти навыки очень помогут мне в будущих проектах.

## Список литературы

....