

# **Лабораторная работа №9**

**Понятие подпрограммы. Отладчик GDB**

Хасанов Тимур

# Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
3	Выполнение задания для самостоятельной работы	24
4	Выводы	37

# Список иллюстраций

2.1	Создание файла lab9-1.asm . . . . .	7
2.2	Запуск Midnight commander . . . . .	8
2.3	Копирование файла in_out.asm в рабочую директорию . . . . .	8
2.4	Вставка кода из файла листинга 9.1 . . . . .	9
2.5	Сборка программы из файла lab9-1.asm и её запуск . . . . .	9
2.6	Изменение файла lab9-1.asm . . . . .	10
2.7	Повторная сборка программы из файла lab9-1.asm и её запуск . .	10
2.8	Создание второго файла: lab9-2.asm . . . . .	10
2.9	Запись кода из листинга 9.2 в файл lab9-2.asm . . . . .	11
2.10	Сборка программы из файла lab9-2.asm . . . . .	11
2.11	Загрузка программы lab9-2.asm в gdb . . . . .	12
2.12	Запуск программы в отладчике . . . . .	12
2.13	Создание брейкпоинта . . . . .	12
2.14	Дизассемблирование программы . . . . .	13
2.15	Переключение на синтаксис intel . . . . .	13
2.16	Повторное дизассемблирование . . . . .	14
2.17	Включение графического отображения кода и выполнения команд	14
2.18	Внешний вид интерфейса . . . . .	14
2.19	включение графического отображения значений регистров и отображение интерфейса . . . . .	15
2.20	Вывод информации о брейкпоинтах . . . . .	15
2.21	Создание брейкпоинта по адресу . . . . .	16
2.22	Повторный вывод информации о брейкпоинтах . . . . .	16
2.23	Выполнение следующей команды в коде программы (1) . . . . .	17
2.24	Выполнение следующей команды в коде программы (2) . . . . .	17
2.25	Выполнение следующей команды в коде программы (3) . . . . .	18
2.26	Выполнение следующей команды в коде программы (4) . . . . .	18
2.27	Выполнение следующей команды в коде программы (5) . . . . .	19
2.28	Вывод значений регистров . . . . .	19
2.29	Значения регистров . . . . .	19
2.30	Вывод значения переменной по имени . . . . .	20
2.31	Вывод значения переменной по адресу . . . . .	20
2.32	Изменение первого символа переменной по имени и вывод переменной . . . . .	20
2.33	Изменение второго символа переменной по адресу и вывод переменной . . . . .	20

2.34	Изменение нескольких символов второй переменной по адресу и вывод переменной . . . . .	21
2.35	Вывод значения регистра в строковом, двоичном и шестнадцатичном виде . . . . .	21
2.36	Изменение значения регистра . . . . .	21
2.37	Завершение работы программы . . . . .	22
2.38	Завершение работы программы (продолжение) . . . . .	22
2.39	Копирование файла из прошлой работы . . . . .	22
2.40	Сборка программы и загрузка в gdb . . . . .	22
2.41	Создание брейкпоинта и запуск программы . . . . .	22
2.42	Вывод значения регистра esp . . . . .	23
2.43	Вывод всех значений в стеке . . . . .	23
3.1	Копирование первого файла самостоятельной работы из прошлой работы . . . . .	24
3.2	Редактирование кода . . . . .	25
3.3	Редактирование кода (продолжение) . . . . .	25
3.4	Сборка и проверка работы программы . . . . .	26
3.5	Создание файла второго задания самостоятельной работы . . . . .	26
3.6	Вставка кода из листинга 9.3 . . . . .	27
3.7	Сборка программы . . . . .	27
3.8	Запуск программы . . . . .	28
3.9	Загрузка программы в gdb . . . . .	28
3.10	Переключение на синтаксис intel . . . . .	28
3.11	Включение графического отображения кода и выполнения команд . . . . .	28
3.12	Включение графического отображения значений регистров . . . . .	28
3.13	Установка брейкпоинта . . . . .	29
3.14	Значение всех регистров на 0 шаге . . . . .	29
3.15	Значение всех регистров на 1 шаге . . . . .	30
3.16	Значение всех регистров на 2 шаге . . . . .	31
3.17	Значение всех регистров на 3 шаге . . . . .	32
3.18	Значение всех регистров на 4 шаге . . . . .	33
3.19	Значение всех регистров на 5 шаге . . . . .	34
3.20	Значение всех регистров на 6 шаге . . . . .	35
3.21	Редактирование кода . . . . .	36
3.22	Сборка кода и проверка выполнения . . . . .	36

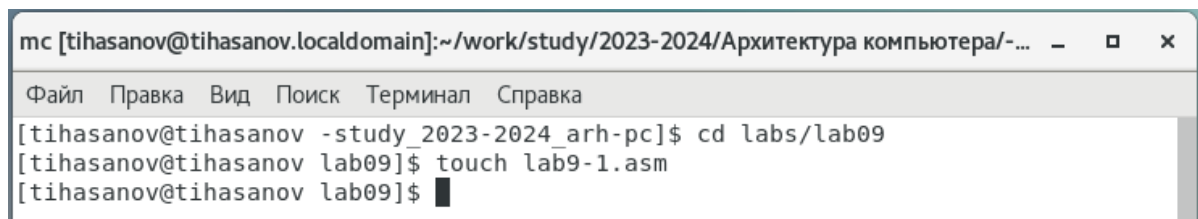
## **Список таблиц**

# 1 Цель работы

Ознакомиться с понятием подпрограмм в Ассемблере и научиться использовать подпрограммы на практике. Ознакомиться с отладчиком gdb и научиться использовать его

## 2 Выполнение лабораторной работы

Для начала выполнения работы необходимо перейти в рабочую папку и создать файл lab9-1.asm (Рис. 2.1):



```
mc [tihasanov@tihasanov.localdomain]:~/work/study/2023-2024/Архитектура компьютера/-... - □ ×
Файл Правка Вид Поиск Терминал Справка
[tihasanov@tihasanov -study_2023-2024_arh-pc]$ cd labs/lab09
[tihasanov@tihasanov lab09]$ touch lab9-1.asm
[tihasanov@tihasanov lab09]$
```

Рис. 2.1: Создание файла lab9-1.asm

Далее, запустим Midnight commander (Рис. 2.2):





Вставим в файл lab9-1.asm код из листинга 9.1 (Рис. 2.4):

```
lab9-1.asm [-M--] 21 L:[ 1+ 0 1/ 35] *(21 / 707b) 0010 0x00A [*][X]
%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
```

Рис. 2.4: Вставка кода из файла листинга 9.1

Соберём программу и посмотрим на вывод (Рис. 2.5):

```
[tihasanov@tihasanov lab09]$ nasm -f elf lab9-1.asm
[tihasanov@tihasanov lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[tihasanov@tihasanov lab09]$ ./lab9-1
Введите x: 10
2x+7=27
[tihasanov@tihasanov lab09]$
```

Рис. 2.5: Сборка программы из файла lab9-1.asm и её запуск

Теперь изменим файл так, чтобы внутри подпрограммы была ещё одна подпрограмма, вычисляющая значение  $g(x)$  и чтобы она передавала значение в первую подпрограмму, которая бы уже вычислила значение  $f(g(x))$  (Рис. 2.6):

```

lab9-1.asm [-M--] 13 L:[ 1+ 7 8/ 42] *(141 / 771)
#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

```

Рис. 2.6: Изменение файла lab9-1.asm

Соберём программку и проверим её работу (Рис. 2.7):

```

[tihasanov@tihasanov lab09]$ nasm -f elf lab9-1.asm
[tihasanov@tihasanov lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[tihasanov@tihasanov lab09]$ ./lab9-1
Введите x: 10
f(g(x))=65
[tihasanov@tihasanov lab09]$ █

```

Рис. 2.7: Повторная сборка программы из файла lab9-1.asm и её запуск

Создадим новый файл (Рис. 2.8):

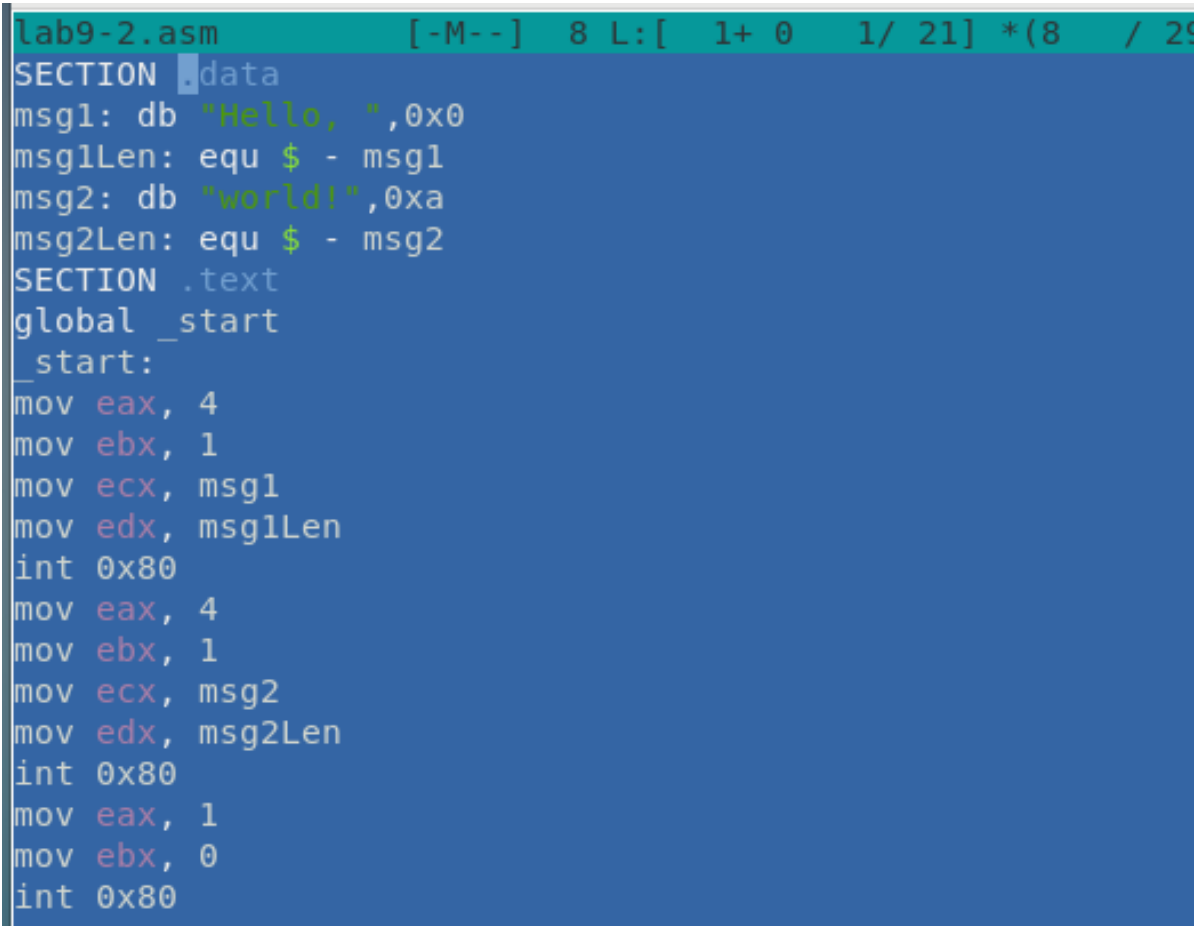
```

[tihasanov@tihasanov lab09]$ touch lab9-2.asm
[tihasanov@tihasanov lab09]$ █

```

Рис. 2.8: Создание второго файла: lab9-2.asm

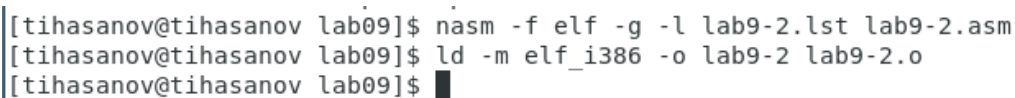
Вставим в него код из листинга 9.2 (Рис. 2.9):



```
lab9-2.asm [-M--] 8 L:[ 1+ 0 1/ 21] *(8 / 25
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. 2.9: Запись кода из листинга 9.2 в файл lab9-2.asm

Соберём программу следующим образом (с использованием аргумента -g) (Рис. 2.10):



```
[tihasanov@tihasanov lab09]$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
[tihasanov@tihasanov lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[tihasanov@tihasanov lab09]$
```

Рис. 2.10: Сборка программы из файла lab9-2.asm

Теперь загрузим её в gdb (Рис. 2.11):

```
[tihasanov@tihasanov lab09]$ gdb lab9-2
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 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".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/tihasanov/work/study/2023-2024/Архитектура компьютера
/-study_2023-2024_arh-pc/labs/lab09/lab9-2...done.
(gdb)
```

Рис. 2.11: Загрузка программы lab9-2.asm в gdb

Запустим её в отладчике с помощью команды run (Рис. 2.12):

```
(gdb) run
Starting program: /home/tihasanov/work/study/2023-2024/Архитектура компьютера/-s
tudy_2023-2024_arh-pc/labs/lab09/lab9-2
Hello, world!
[Inferior 1 (process 5798) exited normally]
(gdb)
```

Рис. 2.12: Запуск программы в отладчике

Создадим брейкпоинт на метке \_start с помощью команды break (Рис. 2.13):

```
(gdb) break _start
Breakpoint 1 at 0x8048080: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/tihasanov/work/study/2023-2024/Архитектура компьютера/-s
tudy_2023-2024_arh-pc/labs/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4
(gdb)
```

Рис. 2.13: Создание брейкпоинта

С помощью команды disassemble дизассемблируем её (Рис. 2.14):

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08048080 <+0>:      mov     $0x4,%eax
    0x08048085 <+5>:      mov     $0x1,%ebx
    0x0804808a <+10>:     mov     $0x80490b8,%ecx
    0x0804808f <+15>:     mov     $0x8,%edx
    0x08048094 <+20>:     int     $0x80
    0x08048096 <+22>:     mov     $0x4,%eax
    0x0804809b <+27>:     mov     $0x1,%ebx
    0x080480a0 <+32>:     mov     $0x80490c0,%ecx
    0x080480a5 <+37>:     mov     $0x7,%edx
    0x080480aa <+42>:     int     $0x80
    0x080480ac <+44>:     mov     $0x1,%eax
    0x080480b1 <+49>:     mov     $0x0,%ebx
    0x080480b6 <+54>:     int     $0x80
End of assembler dump.
(gdb) █
```

---

Рис. 2.14: Дизассемблирование программы

Переключим синтаксис вывода на intel (Рис. 2.15):

```
(gdb) set disassembly-flavor intel
(gdb) █
```

---

Рис. 2.15: Переключение на синтаксис intel

Повторно дизассемблируем программу (Рис. 2.16):

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08048080 <+0>:      mov     eax,0x4
    0x08048085 <+5>:      mov     ebx,0x1
    0x0804808a <+10>:     mov     ecx,0x80490b8
    0x0804808f <+15>:     mov     edx,0x8
    0x08048094 <+20>:     int     0x80
    0x08048096 <+22>:     mov     eax,0x4
    0x0804809b <+27>:     mov     ebx,0x1
    0x080480a0 <+32>:     mov     ecx,0x80490c0
    0x080480a5 <+37>:     mov     edx,0x7
    0x080480aa <+42>:     int     0x80
    0x080480ac <+44>:     mov     eax,0x1
    0x080480b1 <+49>:     mov     ebx,0x0
    0x080480b6 <+54>:     int     0x80
End of assembler dump.
```

Рис. 2.16: Повторное дизассемблирование

Включим графическое отображения кода (Рис. 2.17):

```
(gdb) layout asm
```

Рис. 2.17: Включение графического отображения кода и выполнения команд

Вот как теперь это выглядит (Рис. 2.18):

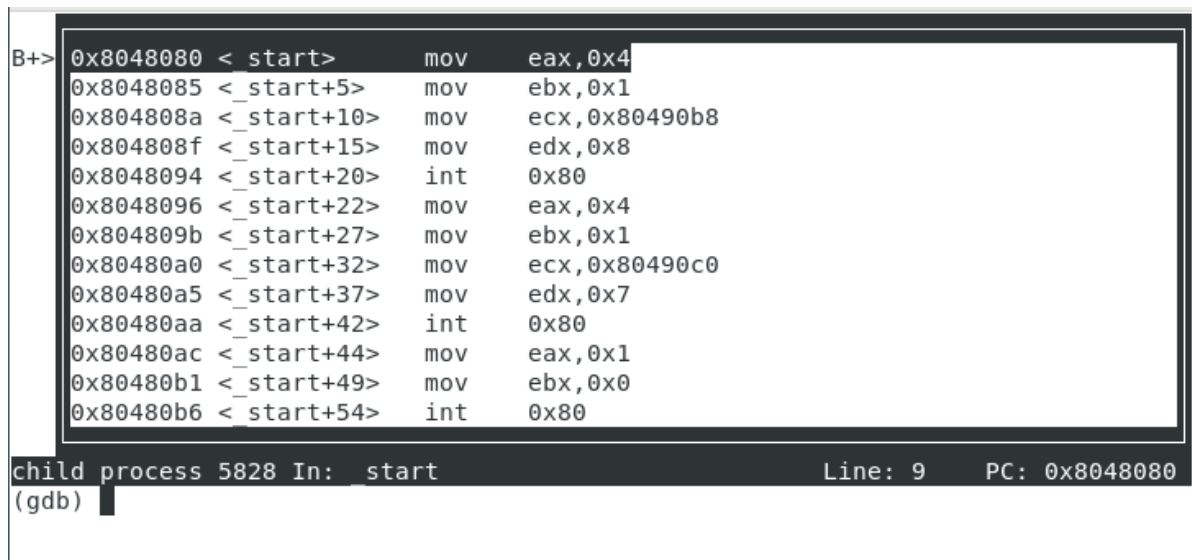


Рис. 2.18: Внешний вид интерфейса

Теперь включим графическое отображение значений регистров (Рис. 2.19):

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcfa0 0xffffcfa0
ebp      0x0      0x0

B+> 0x8048080 < start>    mov     eax,0x4
    0x8048085 <_start+5>   mov     ebx,0x1
    0x804808a <_start+10>  mov     ecx,0x80490b8
    0x804808f <_start+15>  mov     edx,0x8
    0x8048094 <_start+20>  int     0x80
    0x8048096 <_start+22>  mov     eax,0x4

child process 5828 In: start Line: 9 PC: 0x8048080
(gdb)
```

Рис. 2.19: включение графического отображения значений регистров и отображение интерфейса

Выведем информацию о всех брейкпоинтах (Рис. 2.20):

```
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08048080 lab9-2.asm:9
          breakpoint already hit 1 time
(gdb)
```

Рис. 2.20: Вывод информации о брейкпоинтах

Попробуем теперь создать брейкпоинт по адресу (Рис. 2.21):

```

B+> 0x8048080 < start>      mov     eax,0x4
      0x8048085 < _start+5>   mov     ebx,0x1
      0x804808a < _start+10>  mov     ecx,0x80490b8
      0x804808f < _start+15>  mov     edx,0x8
      0x8048094 < _start+20>  int     0x80
      0x8048096 < _start+22>  mov     eax,0x4
      0x804809b < _start+27>  mov     ebx,0x1
      0x80480a0 < _start+32>  mov     ecx,0x80490c0
      0x80480a5 < _start+37>  mov     edx,0x7
      0x80480aa < _start+42>  int     0x80
      0x80480ac < _start+44>  mov     eax,0x1
b+ 0x80480b1 < _start+49>  mov     ebx,0x0
      0x80480b6 < _start+54>  int     0x80

child process 5828 In:  start                               Line: 9      PC: 0x8048080

Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08048080 lab9-2.asm:9
          breakpoint already hit 1 time
(gdb) break *0x80480b1
Breakpoint 2 at 0x80480b1: file lab9-2.asm, line 20.
(gdb) █

```

Рис. 2.21: Создание брейкпоинта по адресу

Повторно выведем информацию о брейкпоинтах (Рис. 2.22):

```

(gdb) i b
Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08048080 lab9-2.asm:9
          breakpoint already hit 1 time
2        breakpoint    keep y  0x080480b1 lab9-2.asm:20
(gdb) █

```

Рис. 2.22: Повторный вывод информации о брейкпоинтах

Теперь 5 раз выполним команду `si` для построчного выполнения кода (Рис. 2.23 - 2.27):



```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcfa0  0xffffcfa0
ebp      0x0      0x0
esi      0x0      0

B+ 0x8048080 < start>      mov     eax,0x4
> 0x8048085 < start+5>     mov     ebx,0x1
0x804808a < _start+10>     mov     ecx,0x80490b8
0x804808f < _start+15>     mov     edx,0x8
0x8048094 < _start+20>     int     0x80
0x8048096 < _start+22>     mov     eax,0x4
0x804809b < _start+27>     mov     ebx,0x1
0x80480a0 < _start+32>     mov     ecx,0x80490c0

Breakpoint 2 at 0x80480b1: file lab9-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y  0x08048080 lab9-2.asm:9
          breakpoint already hit 1 time
2        breakpoint      keep y  0x080480b1 lab9-2.asm:20
(gdb) █

```

Рис. 2.23: Выполнение следующей команды в коде программы (1)

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x1      1
esp      0xffffcfa0  0xffffcfa0
ebp      0x0      0x0
esi      0x0      0

B+ 0x8048080 < _start>      mov     eax,0x4
0x8048085 < start+5>     mov     ebx,0x1
> 0x804808a < start+10>     mov     ecx,0x80490b8
0x804808f < _start+15>     mov     edx,0x8
0x8048094 < _start+20>     int     0x80
0x8048096 < _start+22>     mov     eax,0x4
0x804809b < _start+27>     mov     ebx,0x1
0x80480a0 < _start+32>     mov     ecx,0x80490c0

Breakpoint 2 at 0x80480b1: file lab9-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y  0x08048080 lab9-2.asm:9
          breakpoint already hit 1 time
2        breakpoint      keep y  0x080480b1 lab9-2.asm:20
(gdb) █

```

Рис. 2.24: Выполнение следующей команды в коде программы (2)

```

Register group: general
eax      0x4      4
ecx      0x80490b8  134516920
edx      0x0      0
ebx      0x1      1
esp      0xffffcfa0 0xffffcfa0
ebp      0x0      0x0
esi      0x0      0

B+ 0x8048080 <_start>    mov    eax,0x4
    0x8048085 <_start+5>  mov    ebx,0x1
    0x804808a <_start+10> mov    ecx,0x80490b8
>  0x804808f <_start+15> mov    edx,0x8
    0x8048094 <_start+20> int     0x80
    0x8048096 <_start+22> mov    eax,0x4
    0x804809b <_start+27> mov    ebx,0x1
    0x80480a0 <_start+32> mov    ecx,0x80490c0

Breakpoint 2 at 0x80480b1: file lab9-2.asm, line 20.
(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint     keep y   0x08048080 lab9-2.asm:9
        breakpoint already hit 1 time
2      breakpoint     keep y   0x080480b1 lab9-2.asm:20
(gdb) █

```

Рис. 2.25: Выполнение следующей команды в коде программы (3)

```

Register group: general
eax      0x4      4
ecx      0x80490b8  134516920
edx      0x8      8
ebx      0x1      1
esp      0xffffcfa0 0xffffcfa0
ebp      0x0      0x0
esi      0x0      0

B+ 0x8048080 <_start>    mov    eax,0x4
    0x8048085 <_start+5>  mov    ebx,0x1
    0x804808a <_start+10> mov    ecx,0x80490b8
    0x804808f <_start+15> mov    edx,0x8
>  0x8048094 <_start+20> int     0x80
    0x8048096 <_start+22> mov    eax,0x4
    0x804809b <_start+27> mov    ebx,0x1
    0x80480a0 <_start+32> mov    ecx,0x80490c0

Breakpoint 2 at 0x80480b1: file lab9-2.asm, line 20.
(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint     keep y   0x08048080 lab9-2.asm:9
        breakpoint already hit 1 time
2      breakpoint     keep y   0x080480b1 lab9-2.asm:20
(gdb) █

```

Рис. 2.26: Выполнение следующей команды в коде программы (4)

```

Register group: general
eax      0x8      8
ecx      0x80490b8    134516920
edx      0x8      8Hello,
ebx      0x1      1
esp      0xffffcfa0    0xffffcfa0
ebp      0x0      0x0
esi      0x0      0

B+ 0x8048080 <_start>      mov     eax,0x4
    0x8048085 <_start+5>    mov     ebx,0x1
    0x804808a <_start+10>   mov     ecx,0x80490b8
    0x804808f <_start+15>   mov     edx,0x8
    0x8048094 <_start+20>   int     0x80
> 0x8048096 <_start+22>   mov     eax,0x4
    0x804809b <_start+27>   mov     ebx,0x1
    0x80480a0 <_start+32>   mov     ecx,0x80490c0

Breakpoint 2 at 0x80480b1: file lab9-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08048080 lab9-2.asm:9
          breakpoint already hit 1 time
2        breakpoint      keep y   0x080480b1 lab9-2.asm:20
(gdb) █

```

Рис. 2.27: Выполнение следующей команды в коде программы (5)

Как видим, поменялись значения регистров `eax`, `ecx`, `edx` и `ebx`. Теперь выведем информацию о значениях регистров (Рис. 2.28):

```

(gdb) info registers █

```

Рис. 2.28: Вывод значений регистров

Вот, что нам выводится (Рис. 2.29):

```

eax      0x8      8
ecx      0x80490b8    134516920
edx      0x8      8
ebx      0x1      1
esp      0xffffcfa0    0xffffcfa0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
---Type <return> to continue, or q <return> to quit--- █

```

Рис. 2.29: Значения регистров

Попробуем вывести значения переменной по имени (Рис. 2.30):

```
(gdb) x/lsb &msg1
0x80490b8 <msg1>:      "Hello, "
(gdb) █
```

Рис. 2.30: Вывод значения переменной по имени

Теперь попробуем вывести значения переменной по адресу (Рис. 2.31):

```
(gdb) x/lsb 0x80490c0
0x80490c0 <msg2>:      "world!\n\034"
(gdb) █
```

Рис. 2.31: Вывод значения переменной по адресу

Теперь изменим первый символ переменной (Рис. 2.32):

```
(gdb) set {char}&msg1='h'
(gdb) x/lsb &msg1
0x80490b8 <msg1>:      "hello, "
(gdb) █
```

Рис. 2.32: Изменение первого символа переменной по имени и вывод переменной

А теперь изменим второй символ переменной, уже обратясь по адресу (Рис. 2.33):

```
(gdb) set {char}0x80490b9='h'
(gdb) x/lsb &msg1
0x80490b8 <msg1>:      "hhlllo, "
(gdb) █
```

Рис. 2.33: Изменение второго символа переменной по адресу и вывод переменной

Теперь изменим несколько символов второй переменной (Рис. 2.34):

```
(gdb) set {char}0x80490c0='L'
(gdb) set {char}0x80490c3=' '
(gdb) x/lsb &msg2
0x80490c0 <msg2>:      "Lor d!\n\034"
(gdb) █
```

Рис. 2.34: Изменение нескольких символов второй переменной по адресу и вывод переменной

Теперь попробуем вывести значение регистра в строковом, двоичном и шестнадцатичном виде (Рис. 2.35):

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

Рис. 2.35: Вывод значения регистра в строковом, двоичном и шестнадцатичном виде

Попробуем теперь изменить значение регистра (Рис. 2.36):

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

Рис. 2.36: Изменение значения регистра

Как видим, в регистр записались разные значения. Это связано с тем, что в одном случае мы записываем в него число, а в другом случае - строку. Завершим

работу программы с помощью `continue` (чтобы продолжить выполнение) и выйдем из отладчика (Рис. 2.37 - 2.38):

```
(gdb) continue
Continuing.

Breakpoint 2, _start () at lab9-2.asm:20
(gdb) q
```

Рис. 2.37: Завершение работы программы

```
(gdb) layout asm
[tihasanov@tihasanov lab09]$
```

Рис. 2.38: Завершение работы программы (продолжение)

Скопируем файл из прошлой работы (Рис. 2.39):

```
[tihasanov@tihasanov lab09]$ cp ~/work/study/2023-2024/Архитектура\ компьютера/-study_2023-2024_arh-pc/labs/lab08/lab8-2.asm ~/work/study/2023-2024/Архитектура\ компьютера/-study_2023-2024_arh-pc/labs/lab09/lab9-3.asm
[tihasanov@tihasanov lab09]$
```

Рис. 2.39: Копирование файла из прошлой работы

Соберём его и вгрузим в `gdb` (Рис. 2.40):

```
[tihasanov@tihasanov lab09]$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
[tihasanov@tihasanov lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
[tihasanov@tihasanov lab09]$ gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'
```

Рис. 2.40: Сборка программы и загрузка в `gdb`

Создадим брейкпоинт и запустим программу (Рис. 2.41):

```
(gdb) b _start
Breakpoint 1 at 0x8048148: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/tihasanov/work/study/2023-2024/Архитектура компьютера/-study_2023-2024_arh-pc/labs/lab09/lab9-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 2.41: Создание брейкпоинта и запуск программы

Теперь выведем значение регистра esp, где хранятся данные о стеке (Рис. 2.42):

```
(gdb) x/x $esp
0xffffcf60:      0x00000005
(gdb) █
```

Рис. 2.42: Вывод значения регистра esp

Теперь выведем значение всех элементов стека (Рис. 2.43):

```
(gdb) x/s *(void**)(esp + 4)
0xffffd112:      "/home/tihasanov/work/study/2023-2024/Архитектура компьютера/-study_2023-2024_arh-pc/labs/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd18d:      "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd19f:      "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd1b0:      "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd1b2:      "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:      <Address 0x0 out of bounds>
(gdb) █
```

Рис. 2.43: Вывод всех значений в стеке

Как видим, для вывода каждого элемента стека нам нужно менять значение адреса с шагом 4. Это связано с тем, что именно с шагом 4 располагаются данные в стеке, ведь под каждый элемент выделяется 4 байта

### 3 Выполнение задания для самостоятельной работы

Скопируем файл первого задания прошлой самостоятельной работы (Рис. 3.1):

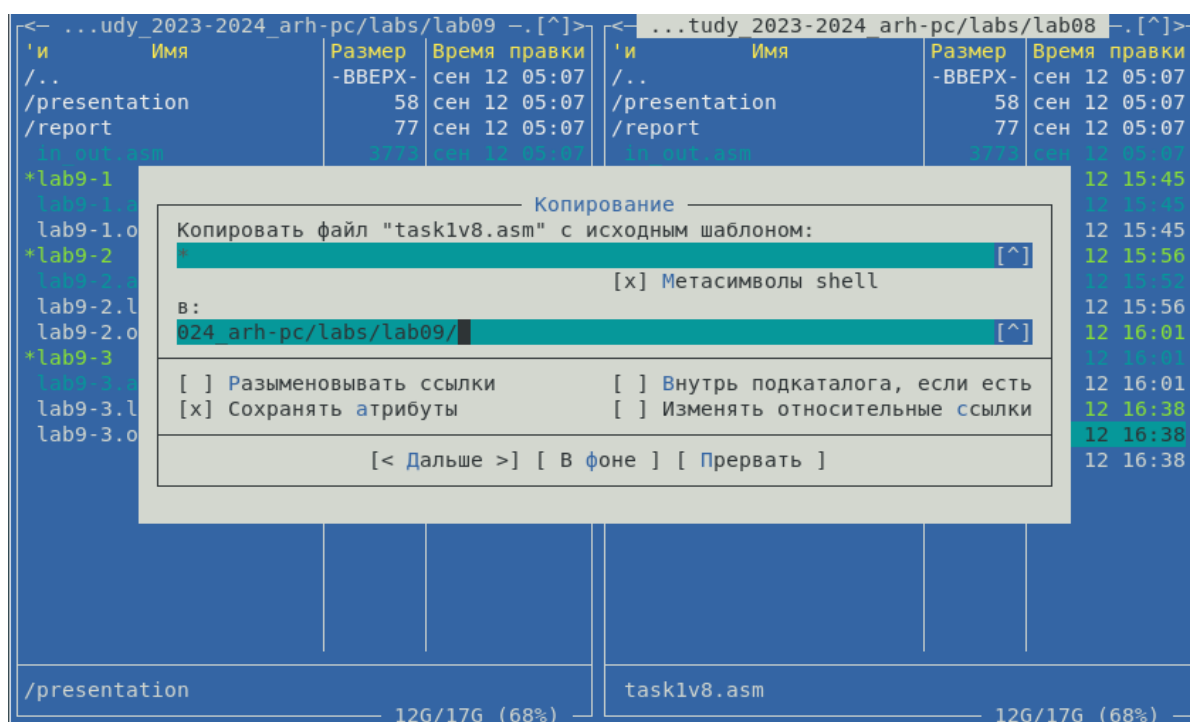


Рис. 3.1: Копирование первого файла самостоятельной работы из прошлой работы

Нам нужно переписать его так, чтобы он использовал для авчисления выражения подпрограмму (Рис. 3.2 - 3.3):



```

task1v8.asm      [-M--] 21 L:[ 1+ 0 1/ 38] *(21 /1751b) 0010 0x00A
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg2 db "Функция: f(x)=7+2x",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
call atoi ; преобразуем символ в число
call _calcul ; вызов подпрограммы для вычисления f(x)
add esi, eax ; добавляем к промежуточной сумме
loop next ; переход к обработке следующего аргумента

```

Рис. 3.2: Редактирование кода

```

_end:
mov eax, msg2
call sprintf
mov eax, msg ; вывод сообщения "Результат: "
call sprintf
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы

_calcul:
mov ebx, 2 ; множитель 2 для f(x) = 7 + 2x
mul ebx ; выполняем умножение `eax = eax * 2`
add eax, 7 ; добавляем 7
ret ; возврат из подпрограммы

```

Рис. 3.3: Редактирование кода (продолжение)

Соберём его и проверим корректность выполнения (Рис. 3.4):

```
[tihasanov@tihasanov lab09]$ nasm -f elf task1v8.asm
[tihasanov@tihasanov lab09]$ ld -m elf_i386 -o task1v8 task1v8.o
[tihasanov@tihasanov lab09]$ ./task1v8 1 2 3 4
Функция:  $f(x)=7+2x$ 
Результат: 48
[tihasanov@tihasanov lab09]$ ./task1v8 1
Функция:  $f(x)=7+2x$ 
Результат: 9
[tihasanov@tihasanov lab09]$ ./task1v8 11 22 33
Функция:  $f(x)=7+2x$ 
Результат: 153
[tihasanov@tihasanov lab09]$ █
```

---

Рис. 3.4: Сборка и проверка работы программы

Создадим файл второго задания самостоятельной работы (Рис. 3.5):

```
[tihasanov@tihasanov lab09]$ touch task2.asm
[tihasanov@tihasanov lab09]$ █
```

---

Рис. 3.5: Создание файла второго задания самостоятельной работы

Вставим в него код из листинга 9.3 (Рис. 3.6):

```

task2.asm          [-M--]  9 L:[  1+19  20/ 20]
#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

```

Рис. 3.6: Вставка кода из листинга 9.3

Соберём его (Рис. 3.7):

```

[tihasanov@tihasanov lab09]$ nasm -f elf -g -l task2.lst task2.asm
[tihasanov@tihasanov lab09]$ ld -m elf_i386 -o task2 task2.o
[tihasanov@tihasanov lab09]$

```

Рис. 3.7: Сборка программы

И запустим (Рис. 3.8):

```
[tihasanov@tihasanov lab09]$ ./task2
Результат: 10
[tihasanov@tihasanov lab09]$
```

Рис. 3.8: Запуск программы

Как видим, код считает значение выражения неправильно. Загрузим его в gdb (Рис. 3.9):

```
[tihasanov@tihasanov lab09]$ gdb task2
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 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".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/tihasanov/work/study/2023-2024/Архитектура компьютера/-study_2023-2024_arh-pc/labs/lab09/task2...done.
(gdb)
```

Рис. 3.9: Вгрузка программы в gdb

Переключим его на синтаксис intel (Рис. 3.10):

```
(gdb) set disassembly-flavor intel
(gdb)
```

Рис. 3.10: Переключение на синтаксис intel

Включим графическое отображение кода (Рис. 3.11):

```
(gdb) layout asm
```

Рис. 3.11: Включение графического отображения кода и выполнения команд

Включим графическое отображение значений регистров (Рис. 3.12):

```
(gdb) layout regs
```

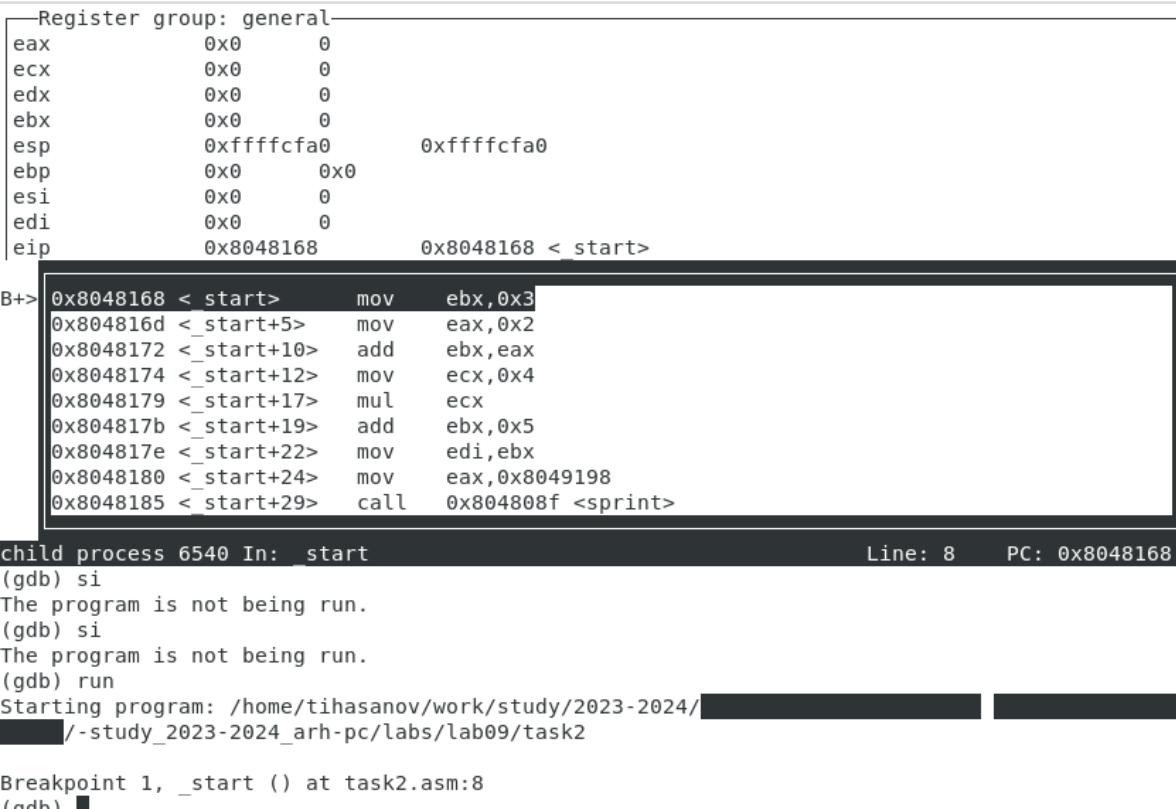
Рис. 3.12: Включение графического отображения значений регистров

Установим брейкпоинт на `_start` (Рис. 3.13):

```
(gdb) break _start
Breakpoint 1 at 0x8048168: file task2.asm, line 8.
(gdb) █
```

Рис. 3.13: Установка брейкпоинта

И начнём построчно выполнять код (Рис. 3.14 - 3.20):



The screenshot shows the GDB interface. At the top, the 'Register group: general' is displayed with the following values:

Register	Value
eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xffffcfa0
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048168

Below the register list, the assembly code for the `_start` function is shown, starting from address `0x8048168`:

```
B+> 0x8048168 <_start> mov ebx,0x3
0x804816d <_start+5> mov eax,0x2
0x8048172 <_start+10> add ebx,eax
0x8048174 <_start+12> mov ecx,0x4
0x8048179 <_start+17> mul ecx
0x804817b <_start+19> add ebx,0x5
0x804817e <_start+22> mov edi,ebx
0x8048180 <_start+24> mov eax,0x8049198
0x8048185 <_start+29> call 0x804808f <sprint>
```

At the bottom, the GDB console shows the following commands and output:

```
child process 6540 In: start Line: 8 PC: 0x8048168
(gdb) si
The program is not being run.
(gdb) si
The program is not being run.
(gdb) run
Starting program: /home/tihasanov/work/study/2023-2024/...
.../study_2023-2024_arh-pc/labs/lab09/task2
Breakpoint 1, _start () at task2.asm:8
(gdb) █
```

Рис. 3.14: Значение всех регистров на 0 шаге

```
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
ebx      0x3      3
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8048168      0x8048168 < start>
eip      0x804816d      0x804816d < start+5>
B+> 0x8048168 < start>      mov     ebx,0x3
B+  0x8048168 < start>      mov     ebx,0x3
> 0x804816d < start+5>      mov     eax,0x2
0x8048174 < start+12>      mov     ecx,0x4
0x8048179 < start+17>      mul     ecx
0x804817b < start+19>      add     ebx,0x5
0x804817e < start+22>      mov     edi,ebx
0x8048180 < start+24>      mov     eax,0x8049198
0x8048185 < start+29>      call    0x804808f <sprint>

child process 6540 In:  start                                     Line: 8      PC: 0x8048168
(gdb) si                                                         9
(gdb) si                                                         d
The program is not being run.
(gdb) run
Starting program: /home/tihasanov/work/study/2023-2024/
/-study_2023-2024_arh-pc/labs/lab09/task2

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

Рис. 3.15: Значение всех регистров на 1 шаге

```

ecx      0x0      0
eax      0x2      2
ebx      0x0      0
ebx      0x3      3
ebx      0x3      3
esi      0x0      0
edi      0x0      0
eip      0x8048168      0x8048168 < start>
eip      0x804816d      0x804816d < _start+5>
Beip     0x8048172      0x8048172 < start+10>
B+ 0x8048168 < start>      mov     ebx,0x3
B+ 0x804816d < _start+5>   mov     eax,0x2
0x804816d < _start+5>     mov     eax,0x2
> 0x8048172 < start+10>    add     ebx,eax
0x804817b < _start+19>    add     ebx,0x5
0x804817e < _start+22>    mov     edi,ebx
0x8048180 < _start+24>    mov     eax,0x8049198
0x8048185 < _start+29>    call    0x804808f <sprint>

child process 6540 In:  start                               Line: 8      PC: 0x8048168
(gdb) si                                                    9
(gdb) si                                                    10      8172
(gdb) run
Starting program: /home/tihasanov/work/study/2023-2024/
/-study_2023-2024_arh-pc/labs/lab09/task2

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

```

Рис. 3.16: Значение всех регистров на 2 шаге

```

eax      0x2      2
eax      0x2      2
ebx      0x3      3
ebx      0x3      3
ebx      0x5      5
edi      0x0      0
eip      0x8048168      0x8048168 < start>
eip      0x804816d      0x804816d < _start+5>
Beip     0x8048172      0x8048172 < _start+10>
Beip     0x8048174      0x8048174 < _start+12>
B+ 0x804816d < _start+5>      mov     eax,0x2
B+ 0x804816d < _start+5>      mov     eax,0x2
> 0x8048172 < _start+10>      add     ebx,eax
> 0x8048172 < _start+10>      add     ebx,eax
> 0x8048174 < _start+12>      mov     ecx,0x4
> 0x8048180 < _start+24>      mov     eax,0x8049198
> 0x8048185 < _start+29>      call    0x804808f <sprint>

child process 6540 In:  _start                               Line: 8      PC: 0x8048168
(gdb) si                                                     9      d
(gdb) si                                                     10     8172
(gdb) run                                                    1      4
/ -study_2023-2024_arh-pc/labs/lab09/task2

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

```

Рис. 3.17: Значение всех регистров на 3 шаге



```

eax      0x2      2
ebx      0x3      3
ecx      0x4      4
ebx      0x5      5
ebx      0x5      5
eip      0x8048168      0x8048168 < start>
eip      0x804816d      0x804816d < _start+5>
Beip     0x8048172      0x8048172 < _start+10>
Beip     0x8048174      0x8048174 < _start+12>
Beip     0x8048179      0x8048179 < _start+17>
B+ 0x804816d < _start+5> mov    eax,0x2
B+ 0x8048172 < _start+10> add    ebx,eax
0x8048172 < _start+10> add    ebx,eax
> 0x8048174 < _start+12> mov    ecx,0x4
0x8048174 < _start+12> mov    ecx,0x4049198
> 0x8048179 < _start+17> mul    ecx,0x40808f < sprint>

child process 6540 In: _start                               Line: 8      PC: 0x8048168
(gdb) si                                                    9
(gdb) si                                                    10      8172
(gdb) run                                                    1
                                                    2
                                                    4
                                                    9
-/study_2023-2024_arh-pc/labs/lab09/task2
Breakpoint 1, _start () at task2.asm:8
(gdb) si
si
(gdb) si
si
(gdb) si
si
(gdb) si
si
(gdb) si
(gdb)

```

Рис. 3.18: Значение всех регистров на 4 шаге

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffcfa0    0xffffcfa0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804817e    0x804817e <_start+22>

B+ 0x8048168 <_start>      mov     ebx,0x3
0x804816d <_start+5>      mov     eax,0x2
0x8048172 <_start+10>     add     ebx,eax
0x8048174 <_start+12>     mov     ecx,0x4
0x8048179 <_start+17>     mul     ecx
0x804817b <_start+19>     add     ebx,0x5
> 0x804817e <_start+22>    mov     edi,ebx
0x8048180 <_start+24>     mov     eax,0x8049198
0x8048185 <_start+29>     call    0x804808f <sprint>

child process 6540 In:  _start                                Line: 13    PC: 0x804817b
info checkpoints -- IDs of currently known checkpoints
info classes -- All Objective-C classes
info common -- Print out the values contained in a Fortran COMMON block
info copying -- Conditions for redistributing copies of GDB
info dcache -- Print information on the dcache performance
info display -- Expressions to display when program stops
info extensions -- All filename extensions associated with a source language
---Type <return> to continue, or q <return> to quit---q
Quit
(gdb) si
(gdb) █

```

Рис. 3.19: Значение всех регистров на 5 шаге

```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa     10
esp      0xffffcfa0 0xffffcfa0
ebp      0x0      0x0
esi      0x0      0
edi      0xa     10
eip      0x8048180 0x8048180 <_start+24>

0x8048179 <_start+17> mul    ecx
0x804817b <_start+19> add    ebx,0x5
0x804817e <_start+22> mov    edi,ebx
> 0x8048180 <_start+24> mov    eax,0x8049198
0x8048185 <_start+29> call  0x804808f <sprint>
0x804818a <_start+34> mov    eax,edi
0x804818c <_start+36> call  0x8048106 <iprintLF>
0x8048191 <_start+41> call  0x804815b <quit>
0x8048196 add    BYTE PTR [eax],al

child process 6540 In: _start Line: 13 PC: 0x804817b
info classes -- All Objective-C classes
info common -- Print out the values contained in a Fortran COMMON block
info copying -- Conditions for redistributing copies of GDB
info dcache -- Print information on the dcache performance
info display -- Expressions to display when program stops
info extensions -- All filename extensions associated with a source language
---Type <return> to continue, or q <return> to quit---q
Quit
(gdb) si
(gdb) si
(gdb) █
```

Рис. 3.20: Значение всех регистров на 6 шаге

Как видим, мы должны были умножить значение регистра `ebx`, но умножили регистр `eax`. Нам необходимо все результаты хранить в регистре `eax`. Изменим код (Рис. 3.21):

```
task2.asm [-M--] 9 L:[ 1+19 20/ 20] *(
#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. 3.21: Редактирование кода

И проверим корректность его выполнения (Рис. 3.22):

```
[tihasanov@tihasanov lab09]$ nasm -f elf -g -l task2.lst task2.asm
[tihasanov@tihasanov lab09]$ ld -m elf_i386 -o task2 task2.o
[tihasanov@tihasanov lab09]$ ./task2
Результат: 25
[tihasanov@tihasanov lab09]$
```

Рис. 3.22: Сборка кода и проверка выполнения

Как видим, теперь код работает корректно

## 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