

Понятие подпрограммы. Отладчик GDB

Лабораторная работа №9

Приходько Иван Иванович

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
3	Выполнение задания для самостоятельной работы	25
4	Выводы	35

Список иллюстраций

2.1	Создание lab9-1.asm	7
2.2	Копирование in_out.asm	7
2.3	Вставка кода из файла листинга 9.1	8
2.4	Сборка и запуск lab9-1.asm	9
2.5	Изменение файла lab9-1.asm	9
2.6	Повторная сборка и запуск lab9-1.asm	10
2.7	Создание lab9-2.asm	10
2.8	Редактирование lab9-2.asm	10
2.9	Сборка lab9-2.asm	11
2.10	Загрузка программы lab9-2.asm в gdb	11
2.11	Создание брейкпоинта	11
2.12	Дизассемблирование программы	12
2.13	Переключение на синтаксис intel	13
2.14	Внешний вид интерфейса	14
2.15	Включение графического отображения значений регистров	15
2.16	Вывод информации о брейкпоинтах	15
2.17	Создание брейкпоинта по адресу	16
2.18	Повторный вывод информации о брейкпоинтах	16
2.19	Выполнение команды в коде программы (1)	17
2.20	Выполнение команды в коде программы (2)	17
2.21	Выполнение команды в коде программы (3)	18
2.22	Выполнение команды в коде программы (4)	18
2.23	Выполнение команды в коде программы (5)	19
2.24	Вывод значений регистров	19
2.25	Значения регистров	20
2.26	Вывод значения переменной по имени	20
2.27	Вывод значения переменной по адресу	20
2.28	Изменение первого символа переменной по имени и вывод переменной	21
2.29	Изменение второго символа переменной по адресу и вывод переменной	21
2.30	Изменение нескольких символов второй переменной по адресу и вывод переменной	21
2.31	Вывод значения регистра в строковом, двоичном и шестнадцатичном виде	22
2.32	Изменение значения регистра	22
2.33	Завершение работы программы	22

2.34	Завершение работы программы	23
2.35	Сборка программы и выгрузка в gdb	23
2.36	Создание брейкпоинта и запуск программы	23
2.37	Вывод значения регистра esp	23
2.38	Вывод всех значений в стеке	24
3.1	Редактирование кода	26
3.2	Сборка и проверка работы программы	27
3.3	Создание файла второго задания самостоятельной работы	27
3.4	Вставка кода из листинга 9.3	28
3.5	Сборка и запуск программы	28
3.6	Выгрузка программы в gdb	29
3.7	Переключение на синтаксис intel	29
3.8	Включение графического отображения кода и выполнения команд	30
3.9	Значение всех регистров на 1 шаге	31
3.10	Значение всех регистров на 2 шаге	31
3.11	Значение всех регистров на 3 шаге	32
3.12	Значение всех регистров на 4 шаге	32
3.13	Значение всех регистров на 5 шаге	33
3.14	Значение всех регистров на 6 шаге	33
3.15	Редактирование кода	34
3.16	Сборка кода и проверка выполнения	34

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 создать файл lab9-1.asm (рис. 2.1).

```
ivanpr1hodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc$ cd ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/
ivanpr1hodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ touch lab9-1.asm
ivanpr1hodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 2.1: Создание lab9-1.asm

Скопируем файл in_out.asm из директории прошлой работы (рис. 2.2).

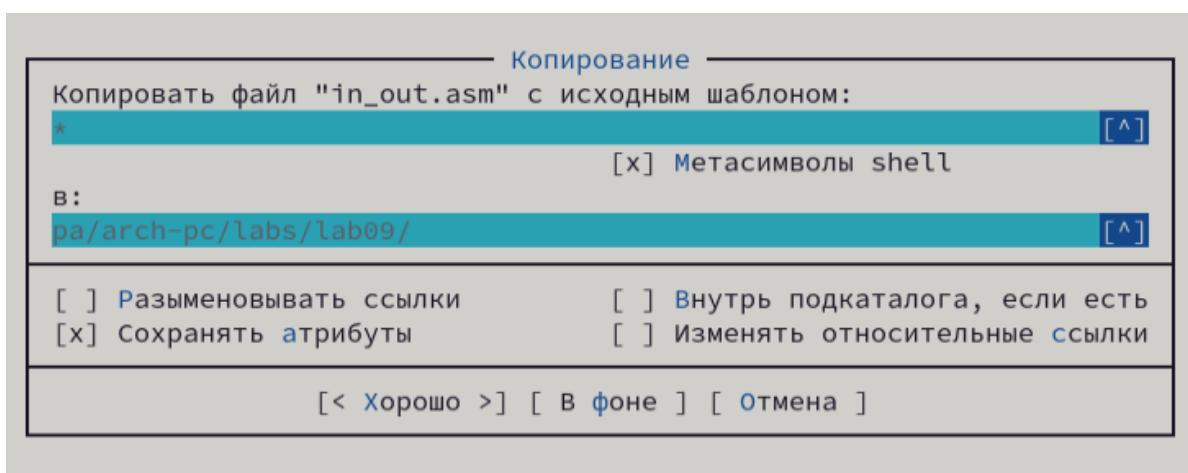


Рис. 2.2: Копирование in_out.asm

Вставим в файл lab9-1.asm код из листинга 9.1 (рис. 2.3).

```

GNU nano 7.2 /
#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. 2.3: Вставка кода из файла листинга 9.1

Соберем и запустим файл (рис. 2.4).

```
ivanprikhodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf lab9-1.asm
ivanprikhodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
ivanprikhodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ./lab9-1
Введите x: 10
2x+7=27
ivanprikhodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 2.4: Сборка и запуск lab9-1.asm

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. 2.5).

```
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

ret ; выход из подпрограммы
```

Рис. 2.5: Изменение файла lab9-1.asm

Повторно соберем и запустим программу (рис. 2.6).

```

ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf lab9-1.asm
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ./lab9-1
Введите x: 1
f(g(x))=11
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ./lab9-1
Введите x: 2
f(g(x))=17
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ./lab9-1
Введите x: 3
f(g(x))=23
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$

```

Рис. 2.6: Повторная сборка и запуск lab9-1.asm

Создадим новый файл и вставим в него код из листинга 9.2 (рис. 2.7).

```

ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ touch lab9-2.asm
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$

```

Рис. 2.7: Создание lab9-2.asm

```

GNU nano 7.2
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. 2.8: Редактирование lab9-2.asm

Соберём программу с использованием аргумента -g (рис. 2.9).

```
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 2.9: Сборка lab9-2.asm

Теперь загрузим её в gdb (рис. 2.10).

```
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 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 lab9-2...
(gdb)
```

Рис. 2.10: Загрузка программы lab9-2.asm в gdb

Создадим брейкпоинт на метке _start с помощью команды break (рис. 2.11).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/ivanprihodko/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/lab9-2

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

Рис. 2.11: Создание брейкпоинта

С помощью команды disassemble дизассемблируем её (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)

```

Рис. 2.12: Дизассемблирование программы

Переключим синтаксис вывода на intel и повторно дизассемблируем программу (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)
```

Рис. 2.13: Переключение на синтаксис intel

Включим графическое отображения кода (рис. 2.14).

```
0x8049132      add     BYTE PTR [eax],al
0x8049134      add     BYTE PTR [eax],al
0x8049136      add     BYTE PTR [eax],al
0x8049138      add     BYTE PTR [eax],al
0x804913a      add     BYTE PTR [eax],al
0x804913c      add     BYTE PTR [eax],al
0x804913e      add     BYTE PTR [eax],al
0x8049140      add     BYTE PTR [eax],al
0x8049142      add     BYTE PTR [eax],al
0x8049144      add     BYTE PTR [eax],al
0x8049146      add     BYTE PTR [eax],al
0x8049148      add     BYTE PTR [eax],al
0x804914a      add     BYTE PTR [eax],al
0x804914c      add     BYTE PTR [eax],al
0x804914e      add     BYTE PTR [eax],al
0x8049150      add     BYTE PTR [eax],al
0x8049152      add     BYTE PTR [eax],al
0x8049154      add     BYTE PTR [eax],al
0x8049156      add     BYTE PTR [eax],al
0x8049158      add     BYTE PTR [eax],al
0x804915a      add     BYTE PTR [eax],al
0x804915c      add     BYTE PTR [eax],al
0x804915e      add     BYTE PTR [eax],al
0x8049160      add     BYTE PTR [eax],al
0x8049162      add     BYTE PTR [eax],al
0x8049164      add     BYTE PTR [eax],al
0x8049166      add     BYTE PTR [eax],al
0x8049168      add     BYTE PTR [eax],al
0x804916a      add     BYTE PTR [eax],al
0x804916c      add     BYTE PTR [eax],al

native process 50691 In: _start
(gdb) 
```

Рис. 2.14: Внешний вид интерфейса

Включим графическое отображение значений регистров (рис. 2.15).

```
[ Register Values Unavailable ]

0x8049132      add     BYTE PTR [eax],al
0x8049134      add     BYTE PTR [eax],al
0x8049136      add     BYTE PTR [eax],al
0x8049138      add     BYTE PTR [eax],al
0x804913a      add     BYTE PTR [eax],al
0x804913c      add     BYTE PTR [eax],al
0x804913e      add     BYTE PTR [eax],al
0x8049140      add     BYTE PTR [eax],al
0x8049142      add     BYTE PTR [eax],al
0x8049144      add     BYTE PTR [eax],al
0x8049146      add     BYTE PTR [eax],al
0x8049148      add     BYTE PTR [eax],al
0x804914a      add     BYTE PTR [eax],al
0x804914c      add     BYTE PTR [eax],al
0x804914e      add     BYTE PTR [eax],al

native process 50691 In: _start
(gdb) layout regs
(gdb)
```

Рис. 2.15: Включение графического отображения значений регистров

Выведем информацию о всех брейкпоинтах (рис. 2.16).

```
(gdb) info breakpoint
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab9-2.asm:9
breakpoint already hit 1 time
(gdb) █
```

Рис. 2.16: Вывод информации о брейкпоинтах

Попробуем теперь создать брейкпоинт по адресу (рис. 2.17).

```
0x804902a <_start+42>  int    0x80
0x804902c <_start+44>  mov    eax,0x1
b+ 0x8049031 <_start+49>  mov    ebx,0x0
0x8049036 <_start+54>  int    0x80

exec No process In:
(gdb) layout regs
(gdb) break 0x8049031
Function "0x8049031" not defined.
Make breakpoint pending on future shared library load? (y
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) █
```

Рис. 2.17: Создание брейкпоинта по адресу

Повторно выведем информацию о брейкпоинтах (рис. 2.18).

```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08049000  lab9-2.asm:9
2        breakpoint      keep y   0x08049031  lab9-2.asm:20
(gdb) █
```

Рис. 2.18: Повторный вывод информации о брейкпоинтах

Теперь 5 раз выполним команду `si` для построчного выполнения кода (рис. 2.19-2.23).


```

Register group: General
eax    0x4      4      ecx    0x0      0      edx    0x0      0
ebx    0x0      0      esp    0xffffcf80  0xffffcf80  ebp    0x0      0x0
esi    0x0      0      edi    0x0      0      eip    0x8049005  0x8049005 <_start+5>
eflags 0x202    [ IF ]  cs     0x23     35      ss     0x2b     43
ds      0x2b     43      es     0x2b     43      fs     0x0      0
gs      0x0      0

B* 0x8049000 <_start+2> mov    eax,ecx
>0x8049005 <_start+5> mov    ebx,0x1
0x8049006 <_start+10> mov    edi,0x8049008
0x804900f <_start+15> mov    edi,0x8
0x8049014 <_start+20> int    0x80
0x8049016 <_start+22> mov    eax,0x4
0x804901b <_start+27> mov    ebx,0x1
0x8049020 <_start+32> mov    ecx,0x8049008
0x8049025 <_start+37> mov    edi,0x7
0x804902a <_start+42> int    0x80
0x804902c <_start+44> mov    eax,0x1
b* 0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80
0x8049038 add    BYTE PTR [eax],al
0x804903a add    BYTE PTR [eax],al

native process 50741 In: _start
1 breakpoint keep y 0x8049000 lab9-2.asm:9
2 breakpoint keep y 0x8049031 lab9-2.asm:20
(gdb) si
The program is not being run.
(gdb) run
Starting program: /home/ivanprithodko/work/study/2024-2025/Архитектура_компьютера/arch-pc/labs/lab09/lab9-2
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.

Breakpoint 1, _start () at lab9-2.asm:9
(gdb) si
(gdb)

```

Рис. 2.19: Выполнение команды в коде программы (1)

```

Register group: General
eax    0x4      4      ecx    0x0      0      edx    0x0      0
ebx    0x1      1      esp    0xffffcf80  0xffffcf80  ebp    0x0      0x0
esi    0x0      0      edi    0x0      0      eip    0x804900a  0x804900a <_start+10>
eflags 0x202    [ IF ]  cs     0x23     35      ss     0x2b     43
ds      0x2b     43      es     0x2b     43      fs     0x0      0
gs      0x0      0

B* 0x8049000 <_start+2> mov    eax,ecx
0x8049005 <_start+5> mov    ebx,0x1
>0x804900a <_start+10> mov    ecx,0x8049008
0x804900f <_start+15> mov    edi,0x8
0x8049014 <_start+20> int    0x80
0x8049016 <_start+22> mov    eax,0x4
0x804901b <_start+27> mov    ebx,0x1
0x8049020 <_start+32> mov    ecx,0x8049008
0x8049025 <_start+37> mov    edi,0x7
0x804902a <_start+42> int    0x80
0x804902c <_start+44> mov    eax,0x1
b* 0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80
0x8049038 add    BYTE PTR [eax],al
0x804903a add    BYTE PTR [eax],al

native process 50741 In: _start
2 breakpoint keep y 0x8049031 lab9-2.asm:20
(gdb) si
The program is not being run.
(gdb) run
Starting program: /home/ivanprithodko/work/study/2024-2025/Архитектура_компьютера/arch-pc/labs/lab09/lab9-2
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.

Breakpoint 1, _start () at lab9-2.asm:9
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.20: Выполнение команды в коде программы (2)

```

~Register group: general
eax      0x4      4      ecx      0x804a000      134520832      edx      0x0      0
ebx      0x1      1      esp      0xffffcf80      0xffffcf80      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0      eip      0x804900f      0x804900f <_start+15>
eflags   0x202    [ IF ]    cs      0x23      35      ss      0x2b      43
ds       0x2b     43      es      0x2b     43      fs      0x0      0
gs       0x0      0

B+ 0x8049000 <_start> mov     eax,0x4
0x8049005 <_start+5> mov     ebx,0x1
0x804900a <_start+10> mov     ecx,0x804a000
>0x804900f <_start+15> mov     edx,0x8
0x8049014 <_start+20> int     0x80
0x8049016 <_start+22> mov     eax,0x4
0x804901b <_start+27> mov     ebx,0x1
0x8049020 <_start+32> mov     ecx,0x804a000
0x8049025 <_start+37> mov     edx,0x7
0x804902a <_start+42> int     0x80
0x804902c <_start+44> mov     eax,0x1
B+ 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54> int     0x80
0x8049038 add     BYTE PTR [eax],al
0x804903a add     BYTE PTR [eax],al

native process 50741 in: _start
(gdb) si
The program is not being run.
(gdb) run
Starting program: /home/ivanprihodko/work/study/2024-2025/Архитектура_компьютера/arch-pc/labs/lab09/lab9-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntuproject.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 lab9-2.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.21: Выполнение команды в коде программы (3)

```

~Register group: general
eax      0x1      4      ecx      0x804a000      134520832      edx      0x0      0
ebx      0x1      1      esp      0xffffcf80      0xffffcf80      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0      eip      0x8049014      0x8049014 <_start+20>
eflags   0x202    [ IF ]    cs      0x23      35      ss      0x2b      43
ds       0x2b     43      es      0x2b     43      fs      0x0      0
gs       0x0      0

B+ 0x8049000 <_start> mov     eax,0x4
0x8049005 <_start+5> mov     ebx,0x1
0x804900a <_start+10> mov     ecx,0x804a000
0x804900f <_start+15> mov     edx,0x8
>0x8049014 <_start+20> int     0x80
0x8049016 <_start+22> mov     eax,0x4
0x804901b <_start+27> mov     ebx,0x1
0x8049020 <_start+32> mov     ecx,0x804a000
0x8049025 <_start+37> mov     edx,0x7
0x804902a <_start+42> int     0x80
0x804902c <_start+44> mov     eax,0x1
B+ 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54> int     0x80
0x8049038 add     BYTE PTR [eax],al
0x804903a add     BYTE PTR [eax],al

native process 50741 in: _start
The program is not being run.
(gdb) run
Starting program: /home/ivanprihodko/work/study/2024-2025/Архитектура_компьютера/arch-pc/labs/lab09/lab9-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntuproject.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 lab9-2.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.22: Выполнение команды в коде программы (4)

```

Register Group: General
eax      0x8      8      ecx      0x804a000      134520832      edx      0x8      8
ebx      0x1      1      esp      0xffffcf80      0xffffcf80      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0      eip      0x8049016      0x8049016 <_start+22>
eflags   0x202      [ IF ]      cs      0x23      35      ss      0x2b      43
ds       0x2b      43      es      0x2b      43      fs       0x0      0
gs       0x0      0

<https://debuginfod.fedoraproject.org>

0* 0x8049000 <_start>      mov     eax,0x4
0x8049005 <_start+5>      mov     ebx,0x1
0x804900a <_start+10>     mov     ecx,0x804a000
0x804900f <_start+15>     mov     edx,0x0
0x8049014 <_start+20>     int     0x0
>0x8049016 <_start+22>     mov     eax,0x4
0x804901b <_start+27>     mov     ebx,0x1
0x8049020 <_start+32>     mov     ecx,0x804a000
0x8049025 <_start+37>     mov     edx,0x7
0x804902a <_start+42>     int     0x0
0x804902c <_start+44>     mov     eax,0x1
0* 0x8049031 <_start+49>     mov     ebx,0x0
0x8049036 <_start+54>     int     0x0
0x8049038      add     BYTE PTR [eax],al
0x804903a      add     BYTE PTR [eax],al

native process 50741 in: _start
(gdb) run
Starting program: /home/ivanprlhodko/work/study/2024-2025/Архитектура_компьютера/arch-pc/labs/lab09/lab9-2

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.

Breakpoint 1, _start () at lab9-2.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.23: Выполнение команды в коде программы (5)

Как видим, поменялись значения регистров eax, ecx, edx и ebx. Теперь выведем информацию о значениях регистров (рис. 2.24).

```
(gdb) info registers
```

Рис. 2.24: Вывод значений регистров

eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xffffcf80	0xffffcf80
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 <_start+22>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x0	0
gs	0x0	0

Рис. 2.25: Значения регистров

Попробуем вывести значение переменной по имени (рис. 2.26).

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

Рис. 2.26: Вывод значения переменной по имени

Теперь попробуем вывести значение переменной по адресу (рис. 2.27).

```
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)
```

Рис. 2.27: Вывод значения переменной по адресу

Теперь изменим первый символ переменной (рис. 2.28).

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

Рис. 2.28: Изменение первого символа переменной по имени и вывод переменной

А теперь изменим второй символ переменной, уже обратившись по адресу (рис. 2.29).

```
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hhlllo, "
(gdb)
```

Рис. 2.29: Изменение второго символа переменной по адресу и вывод переменной

Теперь изменим несколько символов второй переменной (рис. 2.30).

```
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b]' '
Junk after end of expression.
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "Lor d!\n\034"
(gdb)
```

Рис. 2.30: Изменение нескольких символов второй переменной по адресу и вывод переменной

Теперь попробуем вывести значение регистра в строковом, двоичном и шестнадцатичном виде (рис. 2.31).

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

Рис. 2.31: Вывод значения регистра в строковом, двоичном и шестнадцатиричном виде

Попробуем теперь изменить значение регистра (рис. 2.32).

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

Рис. 2.32: 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 (рис. 2.33-2.34).

```
(gdb) continue
Continuing.
Lor d!

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

Рис. 2.33: Завершение работы программы

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

        Inferior 1 [process 50741] will be killed.

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

Рис. 2.34: Завершение работы программы

Скопируем файл из прошлой работы, соберём его и выгрузим в gdb (рис. 2.35).

```
ivanprithodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf -g -i lab9-3.lst lab9-3.asm
ivanprithodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
ld: невозможно найти lab9-3.o: Нет такого файла или каталога
ivanprithodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
ivanprithodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'
```

Рис. 2.35: Сборка программы и выгрузка в gdb

Создадим брейкпоинт и запустим программу (рис. 2.36).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/ivanprithodko/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/lab9-3 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 lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) █
```

Рис. 2.36: Создание брейкпоинта и запуск программы

Теперь выведем значение регистра esp, где хранятся данные о стеке (рис. 2.37).

```
(gdb) x/x $esp
0xffffcf40: 0x00000005
(gdb)
```

Рис. 2.37: Вывод значения регистра esp

Теперь выведем значение всех элементов стека (рис. 2.38).

```
(gdb) x/s *(void**)(esp + 4)
0xffffd101:    "/home/ivanpr1hodko/work/study/2024-2025/Архитектура компьютера/arch-пс/labs/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd16f:    "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd181:    "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd192:    "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd194:    "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb) █
```

Рис. 2.38: 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 и перепишем его так, чтобы он использовал для вычисления выражения подпрограмму (рис. 3.1).

```
GNU nano 7.2 /home/iv
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg2 db "Функция: f(x)=4x-3",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
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_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, 4
mul ebx
sub eax, 3
ret
```

Рис. 3.1: Редактирование кода

Соберем и запустим (рис. 3.2).

```
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf tasklv6.asm
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o tasklv6 tasklv6.o
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ./tasklv6
Функция: f(x)=4x-3
Результат: 0
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ./tasklv6 1 2 3 4
Функция: f(x)=4x-3
Результат: 28
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ./tasklv6 1 2
Функция: f(x)=4x-3
Результат: 6
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ./tasklv6 1 5 2
Функция: f(x)=4x-3
Результат: 23
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 3.2: Сборка и проверка работы программы

Создадим файл второго задания самостоятельной работы (рис. 3.3).

```
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ touch task2.asm
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 3.3: Создание файла второго задания самостоятельной работы

Вставим в него код из листинга 9.3 (рис. 3.4).

```

GNU nano 7.2
#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. 3.4: Вставка кода из листинга 9.3

Соберем его и запустим (рис. 3.5).

```

ivanprikhodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf -g -l task2.lst task2.asm
ivanprikhodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o task2 task2.o
ivanprikhodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ./task2
Результат: 10
ivanprikhodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$

```

Рис. 3.5: Сборка и запуск программы

Как видим, код считает значение выражения неправильно. Загрузим его в gdb (рис. 3.6).

```
ivanprikhodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ gdb task2
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 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 task2...
(gdb) █
```

Рис. 3.6: Выгрузка программы в gdb

Переключим его на синтаксис intel (рис. 3.7).

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

Рис. 3.7: Переключение на синтаксис intel

Включим графическое отображение кода, отображение значений регистров и установим брейкпоинт на `_start` (рис. 3.8).

```
[ Register Values Unavailable ]

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
0x804910c <_start+36>     call    0x8049086 <iprintLF>
0x8049111 <_start+41>     call    0x80490db <quit>

exec No process In:
(gdb) layout regs
(gdb) break _start
Breakpoint 1 at 0x80490e8: file task2.asm, line 8.
(gdb) █
```

Рис. 3.8: Включение графического отображения кода и выполнения команд

И начнём построчно выполнять код (рис. 3.9 - 3.14).

```
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09
--Register group: general--
eax      0x0      0      ecx      0x0      0      edx      0x0      0
ebx      0x3      3      esp      0xffffcf80  0xffffcf80  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0      eip      0x80490ed  0x80490ed <_start+5>
eflags   0x202    [ IF ]  cs      0x23     35      ss      0x2b     43
ds       0x2b     43      es      0x2b     43      fs      0x0      0
gs       0x0      0

B> 0x80490e0 <_start> mov     ebx,ecx
> 0x80490ed <_start+5> mov     eax,0x2
0x80490f2 <_start+10> add     ebx,eax
0x80490f4 <_start+12> mov     ecx,0x4
0x80490f9 <_start+17> mul     ecx
0x80490fd <_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
0x804910c <_start+36> call   0x8049086 <printf>
0x8049111 <_start+41> call   0x80490db <quit>
0x8049116 add     BYTE PTR [eax],al
0x8049118 add     BYTE PTR [eax],al
0x804911a add     BYTE PTR [eax],al

native process 51384 In: _start
(gdb) layout regs
(gdb) break _start
Breakpoint 1 at 0x80490e0: file task2.asm, line 8.
(gdb) run
Starting program: /home/ivanprihodko/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/task2

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 task2.asm:8
(gdb) s1
(gdb) |
```

Рис. 3.9: Значение всех регистров на 1 шаге

```
--Register group: general--
eax      0x2      2      ecx      0x0      0      edx      0x0      0
ebx      0x3      3      esp      0xffffcf80  0xffffcf80  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0      eip      0x80490f2  0x80490f2 <_start+10>
eflags   0x202    [ IF ]  cs      0x23     35      ss      0x2b     43
ds       0x2b     43      es      0x2b     43      fs      0x0      0
gs       0x0      0

B> 0x80490e0 <_start> mov     ebx,ecx
> 0x80490ed <_start+5> mov     eax,0x2
> 0x80490f2 <_start+10> add     ebx,eax
0x80490f4 <_start+12> mov     ecx,0x4
0x80490f9 <_start+17> mul     ecx
0x80490fd <_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
0x804910c <_start+36> call   0x8049086 <printf>
0x8049111 <_start+41> call   0x80490db <quit>
0x8049116 add     BYTE PTR [eax],al
0x8049118 add     BYTE PTR [eax],al
0x804911a add     BYTE PTR [eax],al

native process 51384 In: _start
(gdb) layout regs
(gdb) break _start
Breakpoint 1 at 0x80490e0: file task2.asm, line 8.
(gdb) run
Starting program: /home/ivanprihodko/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/task2

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 task2.asm:8
(gdb) s1
(gdb) s1
(gdb) |
```

Рис. 3.10: Значение всех регистров на 2 шаге

```

Register group: General
eax 0x2 2 ecx 0x0 0 edx 0x0 0
ebx 0x5 5 esp 0xffffcf80 0xffffcf80 ebp 0x0 0
esi 0x0 0 edi 0x0 0 eip 0x80490f4 0x80490f4 <_start+12>
eflags 0x206 [ PF IF ] cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43 fs 0x0 0
gs 0x0 0

0x80490e9 <_start> mov ebx,0x2
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,ebx
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x8049000
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8049080 <printf>
0x8049111 <_start+41> call 0x8049080 <quit>
0x8049116 add BYTE PTR [eax],al
0x8049118 add BYTE PTR [eax],al
0x804911a add BYTE PTR [eax],al

native process 51384 in: _start
(gdb) break _start
Breakpoint 1 at 0x80490e9: file task2.asm, line 8.
(gdb) run
Starting program: /home/ivanprhodko/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/task2

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 task2.asm:8
(gdb) s1
(gdb) s1
(gdb) s1
(gdb)

```

Рис. 3.11: Значение всех регистров на 3 шаге

```

Register group: General
eax 0x2 2 ecx 0x4 4 edx 0x0 0
ebx 0x5 5 esp 0xffffcf80 0xffffcf80 ebp 0x0 0
esi 0x0 0 edi 0x0 0 eip 0x80490f9 0x80490f9 <_start+17>
eflags 0x206 [ PF IF ] cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43 fs 0x0 0
gs 0x0 0

0x80490e9 <_start> mov ebx,0x2
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,ebx
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x8049000
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8049080 <printf>
0x8049111 <_start+41> call 0x8049080 <quit>
0x8049116 add BYTE PTR [eax],al
0x8049118 add BYTE PTR [eax],al
0x804911a add BYTE PTR [eax],al

native process 51384 in: _start
Breakpoint 1 at 0x80490e9: file task2.asm, line 8.
(gdb) run
Starting program: /home/ivanprhodko/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/task2

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 task2.asm:8
(gdb) s1
(gdb) s1
(gdb) s1
(gdb)

```

Рис. 3.12: Значение всех регистров на 4 шаге


```

--Register group: general--
eax      0x0      0      ecx      0x4      4      edx      0x0      0
ebx      0x5      5      esp      0xffffcf80  4      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0      eip      0x80490fb  0x80490fb <_start+19>
eflags   0x206    [ PF IF ]  cs      0x23     35      ss      0x2b     43
ds       0x2b     43      es      0x2b     43      fs      0x0      0
gs       0x0      0

B+ 0x80490e8 <_start> mov ebx,0x3
0x80490ed <_start+5> mov eax,0x2
0x80490f2 <_start+10> add ebx,eax
0x80490f4 <_start+12> mov ecx,0x4
0x80490f5 <_start+17> mul ecx
>0x80490fb <_start+19> add ebx,0x5
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x8040000
0x8049105 <_start+29> call 0x804000f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8040000 <printf>
0x8049111 <_start+41> call 0x8040000 <quit>
0x8049116 add BYTE PTR [eax],al
0x8049118 add BYTE PTR [eax],al
0x804911a add BYTE PTR [eax],al

native process 51384 In: _start L13 PC: 0x80490fb
(gdb) run
Starting program: /home/ivanprilodko/work/study/2024-2025/Архитектура_компьютера/arch-pc/labs/lab09/task2

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 task2.asm:8
(gdb) s1
(gdb) s1
(gdb) s1
(gdb) s1
(gdb) s1
(gdb) s1

```

Рис. 3.13: Значение всех регистров на 5 шаге

```

--Register group: general--
eax      0x0      0      ecx      0x4      4      edx      0x0      0
ebx      0x8      8      esp      0xffffcf80  4      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0      eip      0x80490fe  0x80490fe <_start+22>
eflags   0x206    [ PF IF ]  cs      0x23     35      ss      0x2b     43
ds       0x2b     43      es      0x2b     43      fs      0x0      0
gs       0x0      0

B+ 0x80490e8 <_start> mov ebx,0x3
0x80490ed <_start+5> mov eax,0x2
0x80490f2 <_start+10> add ebx,eax
0x80490f4 <_start+12> mov ecx,0x4
0x80490f5 <_start+17> mul ecx
>0x80490fb <_start+19> add ebx,0x5
>0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x8040000
0x8049105 <_start+29> call 0x804000f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8040000 <printf>
0x8049111 <_start+41> call 0x8040000 <quit>
0x8049116 add BYTE PTR [eax],al
0x8049118 add BYTE PTR [eax],al
0x804911a add BYTE PTR [eax],al

native process 51384 In: _start L14 PC: 0x80490fe
Starting program: /home/ivanprilodko/work/study/2024-2025/Архитектура_компьютера/arch-pc/labs/lab09/task2

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 task2.asm:8
(gdb) s1
(gdb) s1
(gdb) s1
(gdb) s1
(gdb) s1
(gdb) s1

```

Рис. 3.14: 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. 3.15).

```
GNU nano 7.2
#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. 3.15: Редактирование кода

Проверим корректность его выполнения (рис. 3.16).

```
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf -g -l task2.lst task2.asm
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o task2 task2.o
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ./task2
Результат: 25
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 3.16: Сборка кода и проверка выполнения

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.