

Отчёт по лабораторной работе №10

Дисциплина: Архитектура компьютера

Батова Ирина Сергеевна

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
3	Задание для самостоятельной работы	19
4	Выводы	23

Список иллюстраций

2.1	Создание необходимых для работы каталогов и файлов	7
2.2	Ввод листинга 10.1	8
2.3	Запуск программы из файла 'lab10-1.asm'	8
2.4	Добавление еще одной подпрограммы	9
2.5	Запуск измененной программы из файла 'lab10-1.asm'	9
2.6	Создание файла 'lab10-2.asm'	9
2.7	Ввод листинга 10.2	10
2.8	Создание исполняемого файла 10-2	10
2.9	Загрузка файла 10-2 в отладчик	11
2.10	Запуск программы 10-2 в отладчике	11
2.11	Установка первого брейкпоинта	11
2.12	Просмотр дисассимилированного кода	12
2.13	Просмотр intel'овского отображения	12
2.14	Режим псевдографики	13
2.15	Просмотр брейкпоинтов	13
2.16	Установка второго брейкпоинта	13
2.17	Значения регистров	14
2.18	Изменение значений регистров	14
2.19	Просмотр значений регистров	15
2.20	Содержимое переменной msg1	15
2.21	Содержимое переменной msg2	15
2.22	Новое содержимое переменной msg1	15
2.23	Новое содержимое переменной msg2	16
2.24	Значение регистра edx	16
2.25	Изменение значения регистра ebx	16
2.26	Завершение работы в отладчике	16
2.27	Копирование файла	16
2.28	Создание исполняемого файла 10-3	17
2.29	Запуск файла 10-3 в отладчике	17
2.30	Установка брейкпоинта и запуск	17
2.31	Адрес вершины стека	17
2.32	Остальные позиции стека	18
3.1	Редактирование программы	19
3.2	Запуск программы из файла 'lab10-4.asm'	20
3.3	Ввод листинга 10.3	20
3.4	Запуск программы из файла 'lab10-4.asm'	20

3.5	Загрузка программы 10-5 в отладчик	21
3.6	Установка точки останова на метку start	21
3.7	Получение intel'овского отображения	22
3.8	Исправление ошибки	22
3.9	Получение верного результата	22

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

Сначала создаем каталог 'lab10' с помощью команды `mkdir`, переходим в него с помощью команды `cd` и создаем в нем файл 'lab10-1.asm' с помощью команды `touch` (рис. 2.1).

```
[isbatova@fedora ~]$ mkdir ~/work/arch-pc/lab10  
[isbatova@fedora ~]$ cd ~/work/arch-pc/lab10  
[isbatova@fedora lab10]$ touch lab10-1.asm
```

Рис. 2.1: Создание необходимых для работы каталогов и файлов

Открываем файл 'lab10-1.asm' и вводим листинг 10.1 из лабораторной работы (рис. 2.2).

```

1 %include 'in_out.asm'
2
3 SECTION .data
4     msg: DB 'Введите x: ',0
5     result: DB '2x+7= ',0
6
7 SECTION .bss
8     x: RESB 80
9     rez: RESB 80
10
11 SECTION .text
12     global _start
13
14 _start:
15     mov eax, msg
16     call sprint
17
18     mov ecx, x
19     mov edx, 80
20     call sread
21
22     mov eax, x
23     call atoi
24
25     call _calcul
26
27     mov eax, result
28     call sprint
29     mov eax, [rez]
30     call iprintLF
31
32     call quit
33
34 _calcul:
35
36     mov ebx, 2
37     mul ebx
38     add eax, 7
39     mov [rez], eax
40
41     ret

```

Рис. 2.2: Ввод листинга 10.1

Создаем исполняемый файл и запускаем его (рис. 2.3). Программа выводит правильный результат, значит, она написана корректно.

```

[isbatova@fedora lab10]$ nasm -f elf lab10-1.asm
[isbatova@fedora lab10]$ ld -m elf_i386 lab10-1.o -o lab10-1
[isbatova@fedora lab10]$ ./lab10-1
Введите x: 3
2x+7= 13

```

Рис. 2.3: Запуск программы из файла 'lab10-1.asm'

Далее вновь открываем файл 'lab10-1.asm' и редактируем его, добавляя еще одну подпрограмму (для вычисления сложной функции). (рис. 2.4).


```

34 _calcul:
35
36     call _subcalcul
37     mov ebx,2
38     mul ebx
39     add eax,7
40     mov [rez],eax
41
42     ret
43
44 _subcalcul:
45
46     mov ecx,3
47     mul ecx
48     sub eax,1
49     mov [rez],eax
50
51     ret

```

Рис. 2.4: Добавление еще одной подпрограммы

Создаем исполняемый файл и запускаем его (рис. 2.5). Программа выводит правильный результат, значит, она написана корректно.

```

[isbatova@fedora lab10]$ nasm -f elf lab10-1.asm
[isbatova@fedora lab10]$ ld -m elf_i386 lab10-1.o -o lab10-1
[isbatova@fedora lab10]$ ./lab10-1
Введите x: 2
2x+7= 17

```

Рис. 2.5: Запуск измененной программы из файла 'lab10-1.asm'

Для дальнейшей работы создаем файл 'lab10-2.asm' (рис. 2.6).

```

[isbatova@fedora lab10]$ touch lab10-2.asm

```

Рис. 2.6: Создание файла 'lab10-2.asm'

Открываем этот файл 'lab10-2.asm' и вводим листинг 10.2 из лабораторной работы (рис. 2.7).

```

1 SECTION .data
2     msg1: db 'Hello, ',0x0
3     msg1Len: equ $ - msg1
4
5     msg2: db 'world!',0xa
6     msg2Len: equ $ - msg2
7
8 SECTION .text
9     global _start
10
11 _start:
12     mov eax, 4
13     mov ebx, 1
14     mov ecx, msg1
15     mov edx, msg1Len
16     int 0x80
17
18     mov eax, 4
19     mov ebx, 1
20     mov ecx, msg2
21     mov edx, msg2Len
22     int 0x80
23
24     mov eax,1
25     mov ebx,0
26     int 0x80

```

Рис. 2.7: Ввод листинга 10.2

Создаем исполняемый файл, причем трансляцию выполняем с ключом '-g', чтобы мы могли с этим файлом работать в отладчике (рис. 2.8).

```

[isbatova@fedora lab10]$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
[isbatova@fedora lab10]$ ld -m elf_i386 -o lab10-2 lab10-2.o

```

Рис. 2.8: Создание исполняемого файла 10-2

Загружаем файл в отладчик gdb (рис. 2.9).

```
[isbatova@fedora lab10]$ gdb lab10-2
GNU gdb (GDB) Fedora 11.2-3.fc36
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-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 lab10-2...
(gdb) █
```

Рис. 2.9: Загрузка файла 10-2 в отладчик

Для проверки работы программы запускаем ее командой 'run' (рис. 2.10). Видим, что программа работает корректно.

```
(gdb) run
Starting program: /home/isbatova/work/arch-pc/lab10/lab10-2

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.
Hello, world!
[Inferior 1 (process 5047) exited normally]
(gdb) █
```

Рис. 2.10: Запуск программы 10-2 в отладчике

Далее устанавливаем брейкпоинт на метку start и снова запускаем программу (рис. 2.11).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 12.
(gdb) run
Starting program: /home/isbatova/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:12
12      mov eax, 4
(gdb) █
```

Рис. 2.11: Установка первого брейкпоинта

После этого вводим команду 'disassemble _start', чтобы посмотреть дисассемблированный код (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.
```

Рис. 2.12: Просмотр дисассимилированного кода

Далее мы переключаемся на отображение команд с intel'овским синтаксисом (команда 'set disassembly-flavor intel') и вновь просматриваем дисассимилированный код (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.
```

Рис. 2.13: Просмотр intel'овского отображения

Разница между отображениями заключается в удобстве просмотра - в intel'овском синтаксисе отсутствуют знаки '\$' и '%', а также идет сначала регистр, а потом его значение. Это делает отображение более наглядным.

После этого вводим команды 'layout asm' и 'layout regs'. Данные команды включают режим псевдографики, что делает анализ программы удобнее (рис. 2.14).

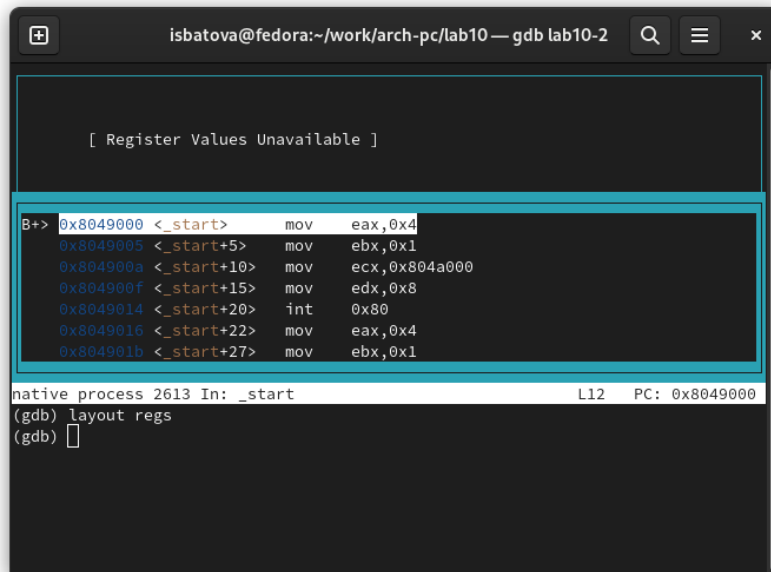


Рис. 2.14: Режим псевдографики

Вводим команду ‘info breakpoints’ для просмотра установленных нами точек останова. Видим одну точку останова, которую мы поставили выше (рис. 2.15).

```
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab10-2.asm:12
breakpoint already hit 1 time
(gdb) □
```

Рис. 2.15: Просмотр брейкпоинтов

После этого нам нужно установить точку останова на предпоследнюю инструкцию. В окне, расположенном посередине, определяем адрес этой инструкции и устанавливаем брейкпоинт по этому адресу. После этого проверяем командой ‘info breakpoints’ наличие двух точек останова (рис. 2.16).

```
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab10-2.asm, line 25.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab10-2.asm:12
breakpoint already hit 1 time
2        breakpoint     keep y   0x08049031 lab10-2.asm:25
(gdb) □
```

Рис. 2.16: Установка второго брейкпоинта

На этом моменте у нас данные значения регистров (рис. 2.17).

```
isbatova@fedora:~/work/arch-pc/lab10 — gdb lab10-2

--Register group: general--
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffd1d0 0      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049000 0      eflags   0x202    [ IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       0x0      0

(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab10-2.asm, line 25.
(gdb) i b
Num  Type  Disp Enb Address  What
1    breakpoint keep y 0x8049031 lab10-2.asm:12
2    breakpoint keep y 0x8049031 lab10-2.asm:25
(gdb) run
Starting program: /home/isbatova/work/arch-pc/lab10/lab10-2
Breakpoint 1, _start () at lab10-2.asm:12
(gdb) 
```

Рис. 2.17: Значения регистров

Далее вводим команду ‘si 5’ и видим, что значения регистров eax, edx и eip меняются (рис. 2.18).

```
isbatova@fedora:~/work/arch-pc/lab10 — gdb lab10-2

--Register group: general--
eax      0x4      8      ecx      0x804a000 134528832
edx      0x7      8      ebx      0x1      1
esp      0xffffd1d0 0      ebp      0x1      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016 0      eflags   0x202    [ IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       0x0      0

(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab10-2.asm, line 25.
(gdb) run
Starting program: /home/isbatova/work/arch-pc/lab10/lab10-2
Breakpoint 1, _start () at lab10-2.asm:12
(gdb) si 5
(gdb) 
```

Рис. 2.18: Изменение значений регистров

Просматриваем значения регистров с помощью команды ‘info registers’ (рис. 2.19).

```

eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1d0 0xffffd1d0
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
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 2.19: Просмотр значений регистров

Далее мы смотрим содержимое переменной 'msg1' (по имени) (рис. 2.20).

```

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

```

Рис. 2.20: Содержимое переменной msg1

После этого с помощью окна посередине определяем адрес переменной 'msg2' и смотрим ее содержимое по адресу (рис. 2.21).

```

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

```

Рис. 2.21: Содержимое переменной msg2

Далее исследуем команду изменения содержимого регистров. Вводим "set {char}&msg1='h' ". Эта команда заменит нам в регистре msg1 первый символ на h. Проверяем этого, просматривая содержимое регистра (рис. 2.22).

```

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

```

Рис. 2.22: Новое содержимое переменной msg1

Аналогично заменяем первый символ в регистре msg2 на 'L' (команда "set {char}&msg2='L' ") и проверяем это (рис. 2.23).

```
(gdb) x/1sb &msg2
0x804a008 <msg2>: "Lor!d!\n\034"
(gdb) □
```

Рис. 2.23: Новое содержимое переменной msg2

Далее мы рассматриваем команду вывода значения регистров. Выводим в трех различных форматах значение регистра edx (рис. 2.24).

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

Рис. 2.24: Значение регистра edx

После этого мы изменяем значение регистра ebx с помощью команды 'set' (рис. 2.25). Мы получаем разные значения, так как во втором случае мы приравниваем регистр к двойке, а в первом вносим значение 2 в регистра.

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

Рис. 2.25: Изменение значения регистра ebx

Далее мы завершаем работу в отладчике и выходим из него (команда 'quit') (рис. 2.26).

```
(gdb) si 2
[Inferior 1 (process 4991) exited normally]
□
```

Рис. 2.26: Завершение работы в отладчике

Для дальнейшей работы копируем файл 'lab9-2.asm' из девятой лабораторной работы в каталог 'lab10' с именем 'lab10-3.asm' (рис. 2.27).

```
[isbatova@fedora lab10]$ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
[isbatova@fedora lab10]$ □
```

Рис. 2.27: Копирование файла

Создаем исполняемый файл (трансляцию выполняем с ключом '-g' (рис. 2.28).

```
[isbatova@fedora lab10]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
[isbatova@fedora lab10]$ ld -m elf_i386 -o lab10-3 lab10-3.o
[isbatova@fedora lab10]$
```

Рис. 2.28: Создание исполняемого файла 10-3

После этого запускаем файл в отладчике с указанием аргументов (рис. 2.29).

```
[isbatova@fedora lab10]$ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora 11.2-3.fc36
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-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 lab10-3...
```

Рис. 2.29: Запуск файла 10-3 в отладчике

Далее устанавливаем брейкпоинт на метку старт и запускаем программу (рис. 2.30).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 7.
(gdb) run
Starting program: /home/isbatova/work/arch-pc/lab10/lab10-3 аргумент1 аргумент

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 lab10-3.asm:7
7      pop ecx
```

Рис. 2.30: Установка брейкпоинта и запуск

Проверяем адрес вершины стека и наличия там 5 позиций (рис. 2.31).

```
(gdb) x/x $esp
0xffffd190: 0x00000005
```

Рис. 2.31: Адрес вершины стека

После этого просматриваем остальные позиции стека (рис. 2.32). Как мы видим, шаг изменения равен четырем. Так происходит потому, что один стек может хранить до четырех байт, и для каждой позиции используется свой стек.

```
(gdb) x/x $esp
0xffffd190:    0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd343:    "/home/isbatova/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd36d:    "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd37f:    "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd390:    "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd392:    "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
```

Рис. 2.32: Остальные позиции стека

3 Задание для самостоятельной работы

1. Для выполнения первого задания копируем файл из заданий для самостоятельной работы из лабораторной работы №9 (я копировала с именем 'lab10-4.asm'). Открываем файл и редактируем его так, чтобы вычисление значений у нас было реализовано как подпрограмма (рис. 3.1).

```
1 %include 'in_out.asm'
2
3 SECTION .data
4     primer: DB 'Функция: f(x)=15x+2',0
5     result: DB 'Результат: ',0
6
7 SECTION .text
8     global _start
9
10 _start:
11     mov eax, primer
12     call sprintf
13
14     pop ecx
15     pop edx
16     sub ecx,1
17     mov esi,0
18
19 next:
20     cmp ecx,0h
21     jz _end
22     pop eax
23     call atoi
24     call _calcul
25     add esi,eax
26     loop next
27
28
29 _end:
30     mov eax,result
31     call sprint
32     mov eax,esi
33     call iprintLF
34     call quit
35
36 _calcul:
37     mov ebx,15
38     mul ebx
39     add eax,2
40     ret
```

Рис. 3.1: Редактирование программы

Сохраняем изменения, создаем исполняемый файл и запускаем его (рис. 3.2). Если посчитать аналитически, получается такой же ответ, поэтому программа работает корректно.

```
[isbatova@fedora lab10]$ nasm -f elf lab10-4.asm
[isbatova@fedora lab10]$ ld -m elf_i386 lab10-4.o -o lab10-4
[isbatova@fedora lab10]$ ./lab10-4 1 2 3 4
Функция: f(x)=15x+2
Результат: 158
```

Рис. 3.2: Запуск программы из файла 'lab10-4.asm'

2. Для начала выполнения второго задания создаем файл 'lab10-5.asm' и вводим в него листинг 10.3 - вычисление выражения $(3+2)*4+5$ (рис. 3.3).

```
1 %include 'in_out.asm'
2
3 SECTION .data
4     div: DB 'Результат: ',0
5
6 SECTION .text
7     global _start
8
9 _start:
10     mov ebx,3
11     mov eax,2
12     add ebx,eax
13     mov ecx,4
14     mul ecx
15     add ebx,5
16     mov edi,ebx
17
18     mov eax,div
19     call sprint
20     mov eax,edi
21     call iprintLF
22     call quit
```

Рис. 3.3: Ввод листинга 10.3

Создаем исполняемый файл и запускаем его (рис. 3.4). Если посчитать аналитически, ответ должен быть 25, а программа выдает 10.

```
[isbatova@fedora lab10]$ nasm -f elf lab10-5.asm
[isbatova@fedora lab10]$ ld -m elf_i386 lab10-5.o -o lab10-5
[isbatova@fedora lab10]$ ./lab10-5
Результат: 10
```

Рис. 3.4: Запуск программы из файла 'lab10-4.asm'

Для поиска ошибки создаем исполняемый файл с ключом '-g' и загружаем файл в отладчик (рис. 3.5).

```
[isbatova@fedora lab10]$ nasm -f elf -g -l lab10-5.lst lab10-5.asm
[isbatova@fedora lab10]$ ld -m elf_i386 -o lab10-5 lab10-5.o
[isbatova@fedora lab10]$ gdb lab10-5
GNU gdb (GDB) Fedora 11.2-3.fc36
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-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 lab10-5...
(gdb) █
```

Рис. 3.5: Загрузка программы 10-5 в отладчик

Устанавливаем точку останова на метку start (рис. 3.6) и дисассемблируем код (с intel'овским изображением (рис. 3.7)).

```
(gdb) break _start
Breakpoint 1 at 0x80490e8: file lab10-5.asm, line 10.
(gdb) run
Starting program: /home/isbatova/work/arch-pc/lab10/lab10-5

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 lab10-5.asm:10
10      mov ebx,3
(gdb) █
```

Рис. 3.6: Установка точки останова на метку start

```
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x080490e8 <+0>:    mov     ebx,0x3
    0x080490ed <+5>:    mov     eax,0x2
    0x080490f2 <+10>:   add     ebx,eax
    0x080490f4 <+12>:   mov     ecx,0x4
    0x080490f9 <+17>:   mul     ecx
    0x080490fb <+19>:   add     ebx,0x5
    0x080490fe <+22>:   mov     edi,ebx
    0x08049100 <+24>:   mov     eax,0x804a000
    0x08049105 <+29>:   call   0x0804900f <sprint>
    0x0804910a <+34>:   mov     eax,edi
    0x0804910c <+36>:   call   0x08049086 <iprintLF>
    0x08049111 <+41>:   call   0x080490db <quit>
End of assembler dump.
(gdb) █
```

Рис. 3.7: Получение intel'овского отображения

Открываем режим псевдографики. Далее, с помощью команды 'si', выполняем программу по одному шагу и на каждом смотрим значения регистров eax и ebx. Путем анализа понимаем, что на 5 шаге у нас на регистр ecx (то есть на 4) умножается не ebx, а eax, поэтому программа и выводит неверный результат. Чтобы исправить это, мы после прибавления к регистру ebx регистра eax записываем получившееся значение в eax. Далее у нас eax умножается на 4, после чего мы обратно записываем значение регистра eax в регистр ebx (рис. 3.8).

```
(gdb) set $eax=$ebx
(gdb) p/s $eax
$3 = 5
(gdb) si
(gdb) si
(gdb) p/s $eax
$4 = 20
(gdb) set $ebx=$eax
(gdb) p/s $ebx
$5 = 20
(gdb) █
```

Рис. 3.8: Исправление ошибки

Далее вводим команду 'continue', чтобы наша программа была выполнена до конца и получаем верный результат (рис. 3.9).

```
$5 = 20
(gdb) c
Continuing.
Результат: 25
[Inferior 1 (process 10386) exited normally]
(gdb) █
```

Рис. 3.9: Получение верного результата

4 Выводы

В данной лабораторной работе мной были приобретены навыки написания программ с использованием подпрограмм, а также навыки работы с методами отладки при помощи GDB.