

Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютеров и операционные системы

Черная София Витальевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
4.1	Реализация подпрограмм в NASM	10
4.2	Отладка программ с помощью GDB	12
4.2.1	Добавление точек останова	16
4.2.2	Работа с данными программы в GDB	17
4.2.3	Обработка аргументов командной строки в GDB	21
4.3	Задания для самостоятельной работы	23
5	Выводы	30
6	Список литературы	31

Список иллюстраций

4.1	Создание файлов для лабораторной работы	10
4.2	Ввод текста программы из листинга 9.1	11
4.3	Запуск исполняемого файла	11
4.4	Изменение текста программы согласно заданию	12
4.5	Запуск исполняемого файла	12
4.6	Ввод текста программы из листинга 9.2	13
4.7	Получение исполняемого файла	13
4.8	Загрузка исполняемого файла в отладчик	14
4.9	Проверка работы файла с помощью команды run	14
4.10	Установка брейкпоинта и запуск программы	14
4.11	Использование команд disassemble и disassembly-flavor intel	15
4.12	Включение режима псевдографики	16
4.13	Установление точек останова и просмотр информации о них	16
4.14	До использования команды stepi	17
4.15	После использования команды stepi	18
4.16	Просмотр значений переменных	19
4.17	Использование команды set	19
4.18	Вывод значения регистра в разных представлениях	20
4.19	Использование команды set для изменения значения регистра	20
4.20	Завершение работы GDB	21
4.21	Создание файла	21
4.22	Загрузка файла с аргументами в отладчик	22
4.23	Установление точки останова и запуск программы	22
4.24	Просмотр значений, введенных в стек	22
4.25	Написание кода подпрограммы	23
4.26	Запуск программы и проверка его вывода	24
4.27	Ввод текста программы из листинга 9.3	25
4.28	Создание и запуск исполняемого файла	26
4.29	Нахождение причины ошибки	26
4.30	Неверное изменение регистра	27
4.31	Исправление ошибки	27
4.32	Ошибка исправлена	28

Список таблиц

1 Цель работы

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

2 Задание

1. Реализация подпрограмм в NASM.
2. Отладка программ с помощью GDB.
3. Добавление точек останова.
4. Работа с данными программы в GDB.
5. Обработка аргументов командной строки в GDB.
6. Задания для самостоятельной работы.

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам.

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки.

Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

Команда `run` (сокращённо `r`) — запускает отлаживаемую программу в оболочке GDB.

Команда `kill` (сокращённо `k`) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. Если в ответ введено `y` (то есть «да»), отладка программы прекращается. Командой `run` её можно начать заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точки

отлова (catchpoints) сохраняются.

Для выхода из отладчика используется команда `quit` (или сокращённо `q`).

Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом `-g`.

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка».

Информацию о всех установленных точках останова можно вывести командой `info` (кратко `i`).

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой `disable`.

Обратно точка останова активируется командой `enable`.

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды `delete`.

Для продолжения остановленной программы используется команда `continue` (`c`). Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число `N`, которое указывает отладчику проигнорировать `N – 1` точку останова (выполнение остановится на `N`-й точке).

Команда `stepi` (кратко `sl`) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию.

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом. Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При

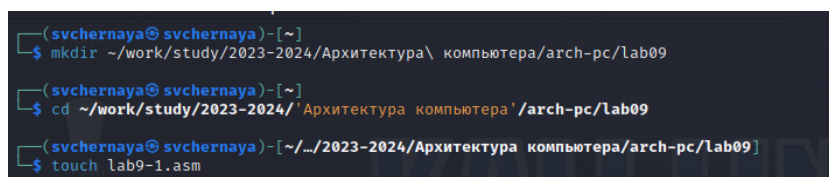
этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `еір` адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы. Подпрограмма завершается инструкцией `ret`, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией `call`, и заносит его в `еір`. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией `call`.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm. (рис. 4.1)



```
(svchernaya@svchernaya)~[~]  
$ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab09  
(svchernaya@svchernaya)~[~]  
$ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab09  
(svchernaya@svchernaya)~[~/2023-2024/Архитектура\ компьютера/arch-pc/lab09]  
$ touch lab9-1.asm
```

Рис. 4.1: Создание файлов для лабораторной работы

Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис. 4.2)

```

~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab9-1.asm - Mousepad
File Edit Search View Document Help
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 mov eax, msg
12 call sprint
13 mov ecx, x
14 mov edx, 80
15 call sread
16 mov eax, x
17 call atoi
18 call _calcul ; Вызов подпрограммы _calcul
19 mov eax, result
20 call sprint
21 mov eax, [res]
22 call iprintLF
23 call quit
24 _calcul:
25 mov ebx, 2
26 mul ebx
27 add eax, 7
28 mov [res], eax
29 ret ; выход из подпрограммы
30

```

Рис. 4.2: Ввод текста программы из листинга 9.1

Создаю исполняемый файл и проверяю его работу. (рис. 4.3)

```

(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ nasm -f elf lab9-1.asm

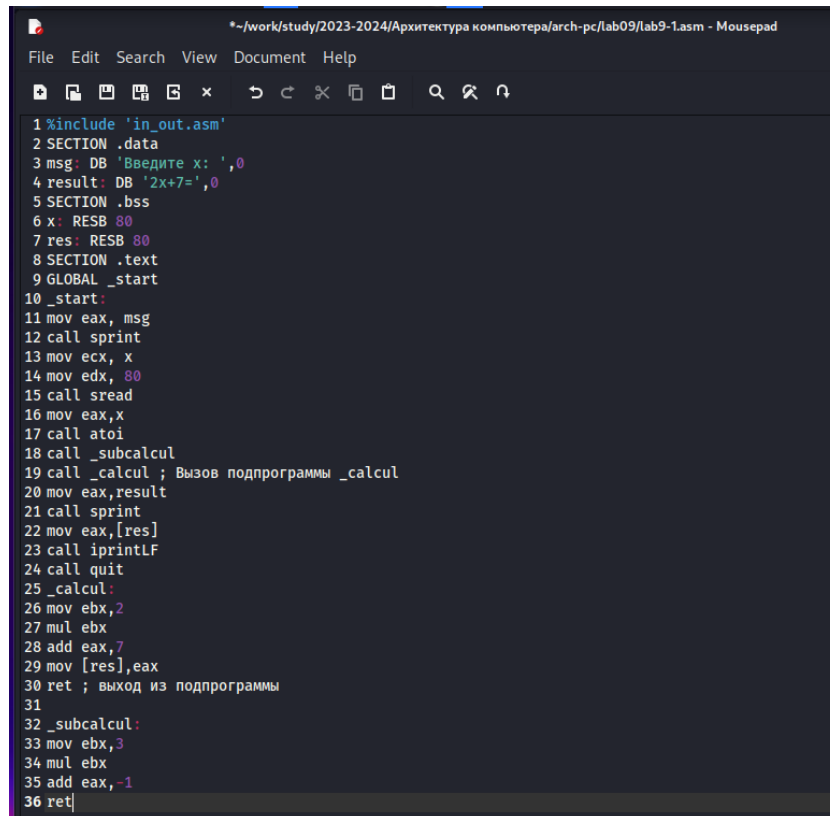
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ ld -m elf_i386 -o lab9-1 lab9-1.o

(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ ./lab9-1
Введите x: 15
2x+7=37

```

Рис. 4.3: Запуск исполняемого файла

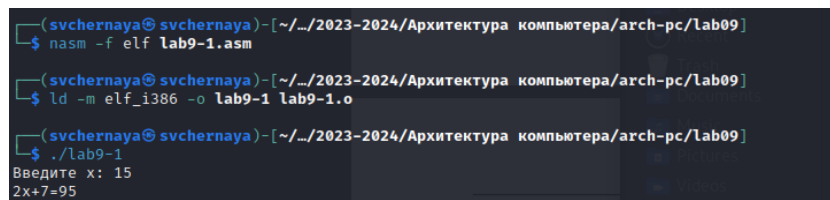
Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul` для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. (рис. 4.4)



```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab9-1.asm - Mousepad
File Edit Search View Document Help
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 mov eax, msg
12 call sprint
13 mov ecx, x
14 mov edx, 80
15 call sread
16 mov eax, x
17 call atoi
18 call _subcalcul
19 call _calcul ; Вызов подпрограммы _calcul
20 mov eax, result
21 call sprint
22 mov eax, [res]
23 call iprintLF
24 call quit
25 _calcul:
26 mov ebx, 2
27 mul ebx
28 add eax, 7
29 mov [res], eax
30 ret ; выход из подпрограммы
31
32 _subcalcul:
33 mov ebx, 3
34 mul ebx
35 add eax, -1
36 ret
```

Рис. 4.4: Изменение текста программы согласно заданию

Создаю исполняемый файл и проверяю его работу. (рис. 4.5)

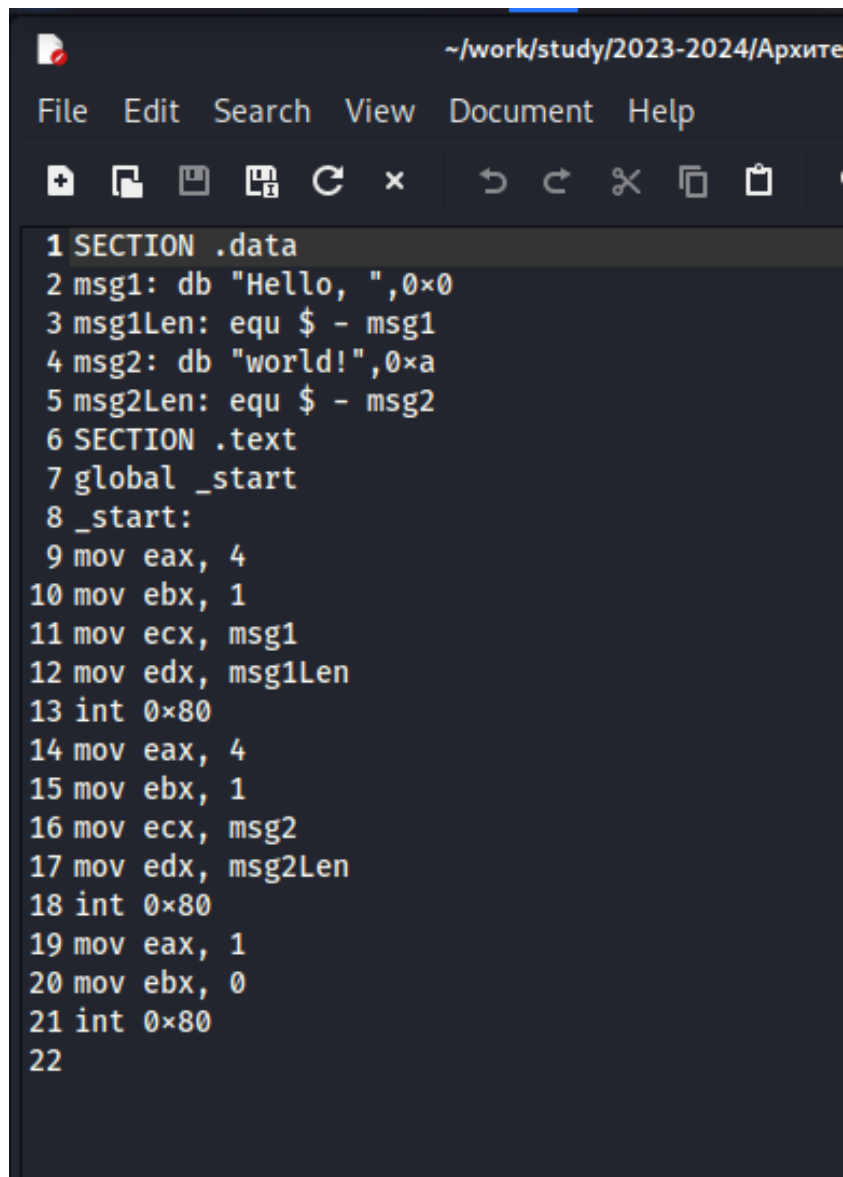


```
(svchernaya@svchernaya) - [~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ nasm -f elf lab9-1.asm
(svchernaya@svchernaya) - [~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ ld -m elf_i386 -o lab9-1 lab9-1.o
(svchernaya@svchernaya) - [~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ ./lab9-1
Введите x: 15
2x+7=95
```

Рис. 4.5: Запуск исполняемого файла

4.2 Отладка программ с помощью GDB

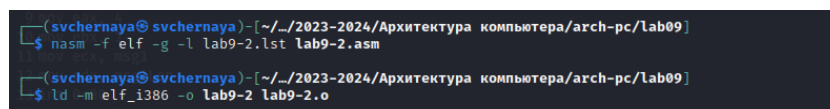
Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (рис. 4.6)



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

Рис. 4.6: Ввод текста программы из листинга 9.2

Получаю исполняемый файл для работы с GDB с ключом '-g'. (рис. 4.7)



```
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ ld -m elf_i386 -o lab9-2 lab9-2.o
```

Рис. 4.7: Получение исполняемого файла

Загружаю исполняемый файл в отладчик gdb. (рис. 4.8)

```
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ gdb lab9-2
GNU gdb (Debian 13.2-1) 13.2
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-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 ...
```

Рис. 4.8: Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run. (рис. 4.9)

```
(gdb) run
Starting program: /home/svchernaya/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 27552) exited normally]
```

Рис. 4.9: Проверка работы файла с помощью команды run

Для более подробного анализа программы устанавливаю брейкпоинт на метку _start и запускаю её. (рис. 4.10)

```
[Inferior 1 (process 27552) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/svchernaya/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab9-2

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

Рис. 4.10: Установка брейкпоинта и запуск программы

Просматриваю дисассемблированный код программы с помощью команды disassemble, начиная с метки _start, и переключаюсь на отображение команд с синтаксисом Intel, введя команду set disassembly-flavor intel. (рис. 4.11)

```

(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) 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. 4.11: Использование команд disassemble и disassembly-flavor intel

В режиме АТТ имена регистров начинаются с символа %, а имена операндов с \$, в то время как в Intel используется привычный нам синтаксис.

Включая режим псевдографики для более удобного анализа программы с помощью команд layout asm и layout regs. (рис. 4.12)



Рис. 4.12: Включение режима псевдографики

4.2.1 Добавление точек останова

Проверяю, что точка останова по имени метки `_start` установлена с помощью команды `info breakpoints` и устанавливаю еще одну точку останова по адресу инструкции `mov ebx,0x0`. Просматриваю информацию о всех установленных точках останова. (рис. 4.13)

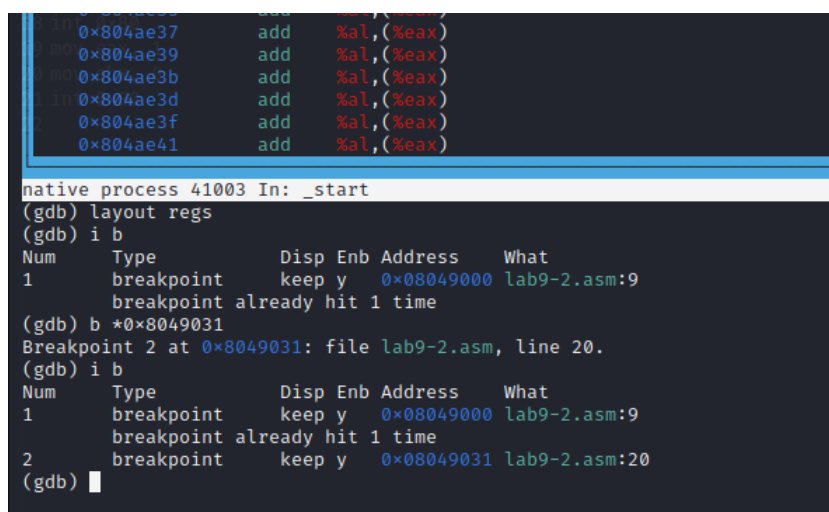
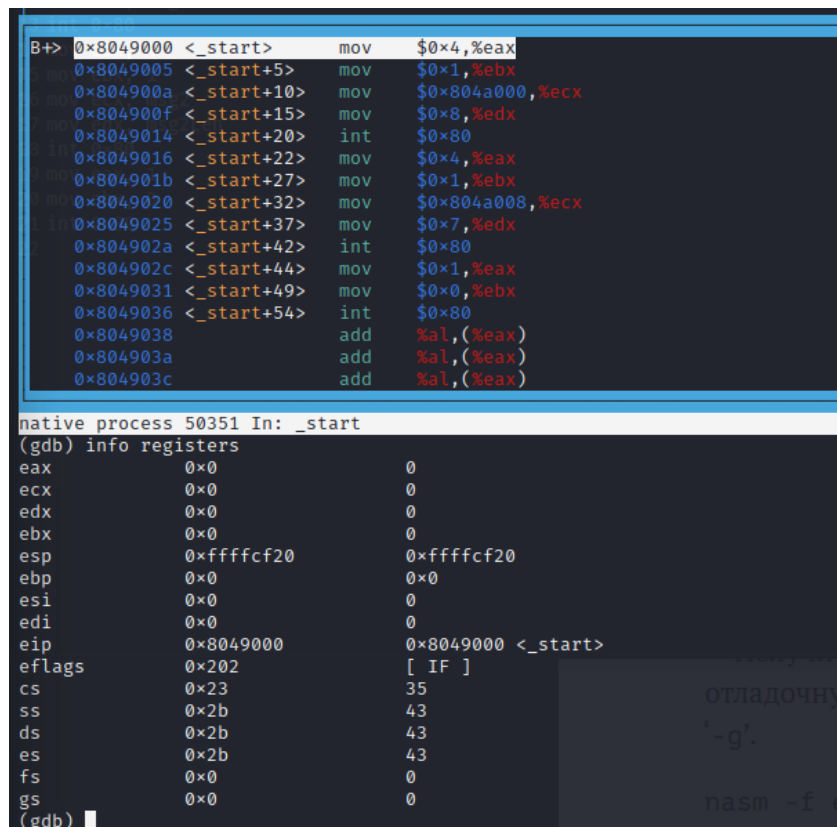


Рис. 4.13: Установление точек останова и просмотр информации о них

4.2.2 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды `stepi` и слежу за изменением значений регистров. (рис. 4.14)



The screenshot shows the GDB interface with two main panels. The top panel displays assembly instructions for a native process. The bottom panel shows the current state of the registers.

```
B+> 0x8049000 <_start>      mov     $0x4,%eax
0x8049005 <_start+5>        mov     $0x1,%ebx
0x804900a <_start+10>       mov     $0x804a000,%ecx
0x804900f <_start+15>       mov     $0x8,%edx
0x8049014 <_start+20>       int     $0x80
0x8049016 <_start+22>       mov     $0x4,%eax
0x804901b <_start+27>       mov     $0x1,%ebx
0x8049020 <_start+32>       mov     $0x804a008,%ecx
0x8049025 <_start+37>       mov     $0x7,%edx
0x804902a <_start+42>       int     $0x80
0x804902c <_start+44>       mov     $0x1,%eax
0x8049031 <_start+49>       mov     $0x0,%ebx
0x8049036 <_start+54>       int     $0x80
0x8049038                add     %al,(%eax)
0x804903a                add     %al,(%eax)
0x804903c                add     %al,(%eax)
```

```
native process 50351 In: _start
(gdb) info registers
eax             0x0             0
ecx             0x0             0
edx             0x0             0
ebx             0x0             0
esp             0xffffcf20      0xffffcf20
ebp             0x0             0x0
esi             0x0             0
edi             0x0             0
eip             0x8049000      0x8049000 <_start>
eflags         0x202           [ IF ]
cs              0x23           35
ss              0x2b           43
ds              0x2b           43
es              0x2b           43
fs              0x0             0
gs              0x0             0
(gdb)
```

Рис. 4.14: До использования команды `stepi`

(рис. 4.15)

```

svchernaya@svchernaya: ~/work/study/2023-2024/Архитектура компьюте
File Actions Edit View Help
Register group: general
eax 0x8 8 ecx 0x80
edx 0x8 8 ebx 0x1
esp 0xffffcf20 0xffffcf20 ebp 0x0
esi 0x0 0 edi 0x0
eip 0x8049016 0x8049016 <_start+22> eflags 0x20
cs 0x23 35 ss 0x2b
ds 0x2b 43 es 0x2b
fs 0x0 0 gs 0x0
global _start
_start:
    mov eax, 4
    mov ebx, 1
    mov ecx, msg1
    mov edx, msglen
B+ 0x8049000 <_start> mov $0x4,%eax
    0x8049005 <_start+5> mov $0x1,%ebx
    0x804900a <_start+10> mov $0x804a000,%ecx
    0x804900f <_start+15> mov $0x8,%edx
    0x8049014 <_start+20> int $0x80
> 0x8049016 <_start+22> mov $0x4,%eax
    0x804901b <_start+27> mov $0x1,%ebx
    0x8049020 <_start+32> mov $0x804a008,%ecx
    0x8049025 <_start+37> mov $0x7,%edx
    0x804902a <_start+42> int $0x80
    0x804902c <_start+44> mov $0x1,%eax
    0x8049031 <_start+49> mov $0x0,%ebx
    0x8049036 <_start+54> int $0x80
    0x8049038 add %al,%eax
    0x804903a add %al,%eax
    0x804903c add %al,%eax
native process 50351 In: _start
(gdb) info registers
eax 0x8 8
ecx 0x804a000 134520832
edx 0x8 8
ebx 0x1 1
esp 0xffffcf20 0xffffcf20
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. 4.15: После использования команды stepi

Изменились значения регистров eax, ecx, edx и ebx.

Просматриваю значение переменной msg1 по имени с помощью команды x/1sb &msg1 и значение переменной msg2 по ее адресу. (рис. 4.16)

```

svchernaya@svchernaya: ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09
File Actions Edit View Help
Register group: general
eax 0x8 8 ecx 0x804a000 134
edx 0x8 8 ebx 0x1 1
esp 0xffffcf20 0xffffcf20 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

global _start
_start:
    mov eax, 4
    mov ebx, 1
    mov ecx, msg1
    mov edx, msglen

0x8049000 <_start> mov $0x4,%eax
0x8049005 <_start+5> mov $0x1,%ebx
0x804900a <_start+10> mov $0x804a000,%ecx
0x804900f <_start+15> mov $0x8,%edx
0x8049014 <_start+20> int $0x80
> 0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27> mov $0x1,%ebx
0x8049020 <_start+32> mov $0x804a008,%ecx
0x8049025 <_start+37> mov $0x7,%edx
0x804902a <_start+42> int $0x80
0x804902c <_start+44> mov $0x1,%eax
0x8049031 <_start+49> mov $0x0,%ebx
0x8049036 <_start+54> int $0x80
0x8049038 add %al,(%eax)
0x804903a add %al,(%eax)
0x804903c add %al,(%eax)

native process 50351 In: _start
ebx 0x1 1
esp 0xffffcf20 0xffffcf20
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
(gdb) x/1sb 0msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 4.16: Просмотр значений переменных

С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2. (рис. 4.17)

```

No symbol 'msg1' in current context.
(gdb) set {char}0msg1='h'
(gdb) x/1sb 0msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}0msg2='b'
(gdb) x/1sb 0msg2
0x804a008 <msg2>: "borld!\n\034"
(gdb)

```

Рис. 4.17: Использование команды set

Вывожу в шестнадцатеричном формате, в двоичном формате и в символьном

виде соответственно значение регистра `edx` с помощью команды `print p/F $val`.
(рис. 4.18)

```
$8 = 0x8
(gdb) p/t $edx
$9 = 1000
(gdb) p/c $edx
$10 = 8 '\b'
(gdb) █
```

Рис. 4.18: Вывод значения регистра в разных представлениях

С помощью команды `set` изменяю значение регистра `ebx` в соответствии с заданием. (рис. 4.19)

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

Рис. 4.19: Использование команды `set` для изменения значения регистра

Разница вывода команд `p/s $ebx` отличается тем, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется.

Завершаю выполнение программы с помощью команды `continue` и выхожу из GDB с помощью команды `quit`. (рис. 4.20)

The screenshot shows the GDB interface with the following content:

Register group: general

eax	0x1	1
ecx	0x804a008	134520840
edx	0x7	7
ebx	0x1	1
esp	0xffffcf20	0xffffcf20
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049031	0x8049031 <_start+49>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43

Assembly code snippet:

```

0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27> mov $0x1,%ebx
0x8049020 <_start+32> mov $0x804a008,%ecx
0x8049025 <_start+37> mov $0x7,%edx
0x804902a <_start+42> int $0x80
0x804902c <_start+44> mov $0x1,%eax
B> 0x8049031 <_start+49> mov $0x0,%ebx
0x8049036 <_start+54> int $0x80
0x8049038 add %al,(%eax)
0x804903a add %al,(%eax)
0x804903c add %al,(%eax)
0x804903e add %al,(%eax)

```

GDB session log:

```

native process 55219 In: _start L20 PC: 0x8049031
$3 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$4 = 2
(gdb) c
Continuing.
world!
Breakpoint 2, _start () at lab9-2.asm:20
(gdb) q
A debugging session is active.
Inferior 1 [process 55219] will be killed.
Quit anyway? (y or n)

```

Рис. 4.20: Завершение работы GDB

4.2.3 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09-3.asm и создаю исполняемый файл. (рис. 4.21)

The screenshot shows a terminal window with the following commands and output:

```

(svchernaya@svchernaya)~[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ cp ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab08/lab8-2.asm ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab09/lab9-3.asm
(svchernaya@svchernaya)~[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
(svchernaya@svchernaya)~[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ ld -m elf_i386 -o lab9-3 lab9-3.o
(svchernaya@svchernaya)~[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$

```

Рис. 4.21: Создание файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргумен-

ты с использованием ключа `-args`. (рис. 4.22)

```
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Debian 13.2-1) 13.2
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-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 lab9-3...
(gdb)
```

Рис. 4.22: Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. 4.23)

```
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Debian 13.2-1) 13.2
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-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 lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/svchernaya/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab9-3 аргумент1 аргумент 2 аргумент\ 3
Breakpoint 1, _start () at lab9-3.asm:5
5      pop еск ; Извлекаем из стека в 'еск' количество
(gdb)
```

Рис. 4.23: Установление точки останова и запуск программы

Посматриваю вершину стека и позиции стека по их адресам. (рис. 4.24)

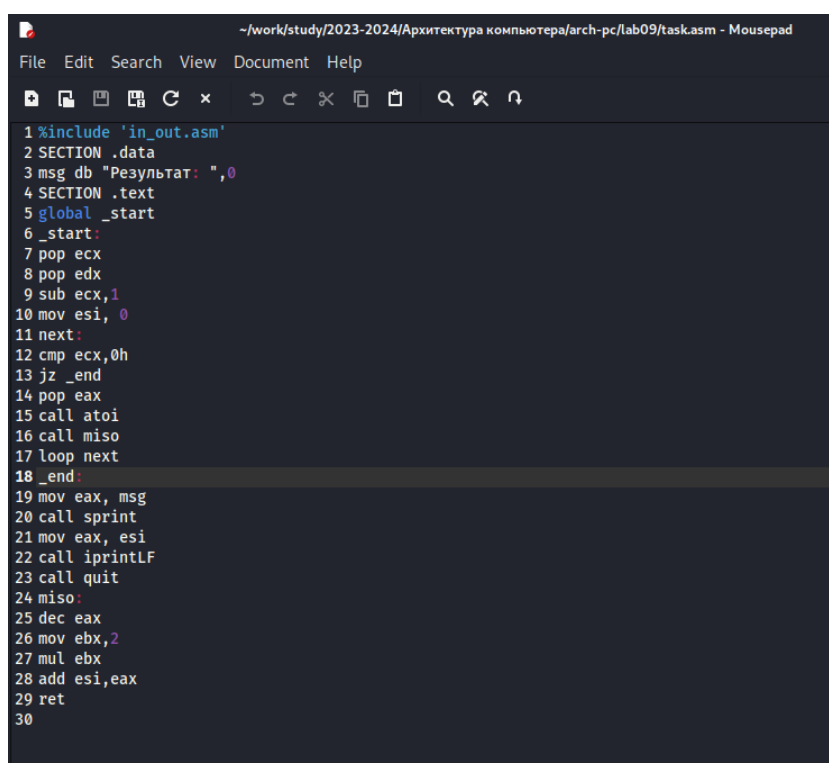
```
(gdb) pop еск ; Извлекаем из стека в 'еск' количество
(gdb) x/x $esp
0xffffced0: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd0bd: "/home/svchernaya/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd124: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd136: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd147: "2"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.24: Просмотр значений, введенных в стек

Шаг изменения адреса равен 4, т.к количество аргументов командной строки равно 4.

4.3 Задания для самостоятельной работы

1. Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. (рис. 4.25)



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx
8 pop edx
9 sub ecx,1
10 mov esi, 0
11 next:
12 cmp ecx,0h
13 jz _end
14 pop eax
15 call atoi
16 call miso
17 loop next
18 _end:
19 mov eax, msg
20 call sprint
21 mov eax, esi
22 call iprintLF
23 call quit
24 miso:
25 dec eax
26 mov ebx,2
27 mul ebx
28 add esi,eax
29 ret
30
```

Рис. 4.25: Написание кода подпрограммы

Запускаю код и проверяю, что она работает корректно. (рис. 4.26)

```
(svchernaya@svchernaya)-[~/./2023-2024/Архитектура компьютера/arch-pc/lab09]
$ nasm -f elf task.asm
(svchernaya@svchernaya)-[~/./2023-2024/Архитектура компьютера/arch-pc/lab09]
$ ld -m elf_i386 -o task task.o
(svchernaya@svchernaya)-[~/./2023-2024/Архитектура компьютера/arch-pc/lab09]
$ ./task 1 2 3
Результат: 6
```

Рис. 4.26: Запуск программы и проверка его вывода

Код программы:

```
%include 'in_out.asm'

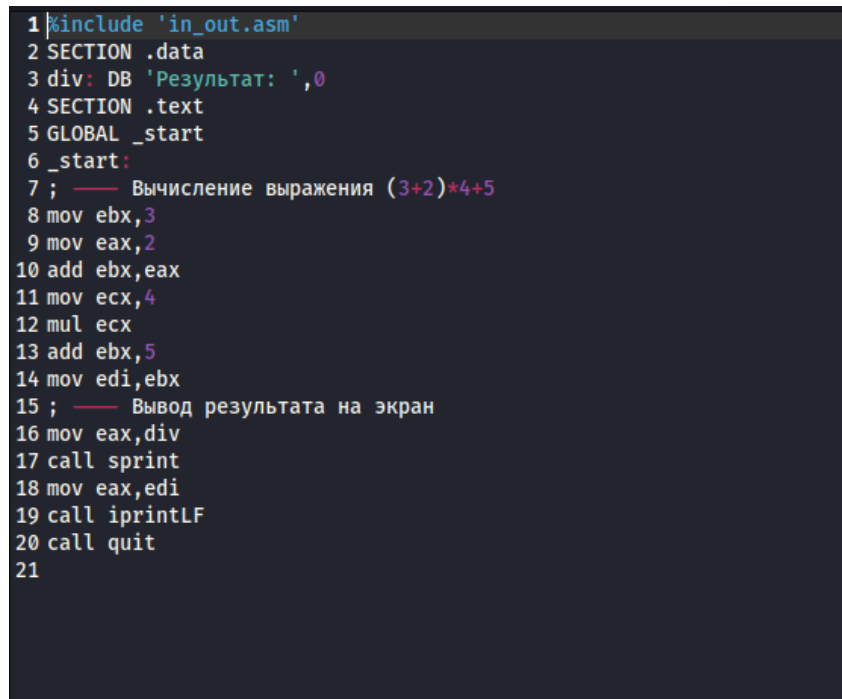
SECTION .data
msg db "Результат: ",0

SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0
next:
cmp ecx,0h
jz _end
pop eax
call atoi
call miso
loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```



```
miso:
dec eax
mov ebx,2
mul ebx
add esi,eax
ret
```

2. Ввожу в файл task1.asm текст программы из листинга 9.3. (рис. 4.27)



```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ——— Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,ebx
15 ; ——— Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
21
```

Рис. 4.27: Ввод текста программы из листинга 9.3

При корректной работе программы должно выводиться “25”. Создаю исполняемый файл и запускаю его. (рис. 4.28)

```
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ touch task1.asm

(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ nasm -f elf task1.asm

(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ ld -m elf_i386 -o task1 task1.o

(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab09]
$ ./task1
Результат: 10
```

Рис. 4.28: Создание и запуск исполняемого файла

Видим, что в выводе мы получаем неправильный ответ.

Получаю исполняемый файл для работы с GDB, запускаю его и ставлю брейкпоинты для каждой инструкции, связанной с вычислениями. С помощью команды `continue` прохожусь по каждому брейкпоинту и слежу за изменениями значений регистров.

При выполнении инструкции `mul ecx` происходит умножение `ecx` на `eax`, то есть 4 на 2, вместо умножения 4 на 5 (регистр `ebx`). Происходит это из-за того, что стоящая перед `mov ecx,4` инструкция `add ebx,ebx` не связана с `mul ecx`, но связана инструкция `mov eax,2`. (рис. 4.29)

```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd160 0xffffd160
ebp      0x0      0x0

B+ 0x80490f9 <_start+17> mul    ecx
B+ 0x80490fb <_start+19> add    ebx,ebx
B+ 0x80490fe <_start+22> mov    edi,ebx
0x8049100 <_start+24> mov    eax,0x804a000
0x8049105 <_start+29> call   0x804900f <sprint>

native process 14573 In: _start L14 PC: 0x80490fe
Continuing.

Breakpoint 6, _start () at task2.asm:13
(gdb) c
Continuing.

Breakpoint 7, _start () at task2.asm:14
(gdb)
```

Рис. 4.29: Нахождение причины ошибки

Из-за этого мы получаем неправильный ответ. (рис. 4.30)

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd160 0xffffd160
ebp      0x0      0x0

B+ 0x80490f9 <_start+17> mul    ecx
B+ 0x80490fb <_start+19> add    ebx,0x5
B+ 0x80490fe <_start+22> mov    edi,ebx
0x8049100 <_start+24> mov    eax,0x804a000
0x8049105 <_start+29> call   0x804900f <sprint>

native process 14573 In: _start L14 PC: 0x80490fe
Continuing.

Breakpoint 6, _start () at task2.asm:13
(gdb) c
Continuing.

Breakpoint 7, _start () at task2.asm:14
(gdb)

```

Рис. 4.30: Неверное изменение регистра

Исправляем ошибку, добавляя после `add ebx,eax` `mov eax,ebx` и заменяя `ebx` на `eax` в инструкциях `add ebx,5` и `mov edi,ebx`. (рис. 4.31)

```

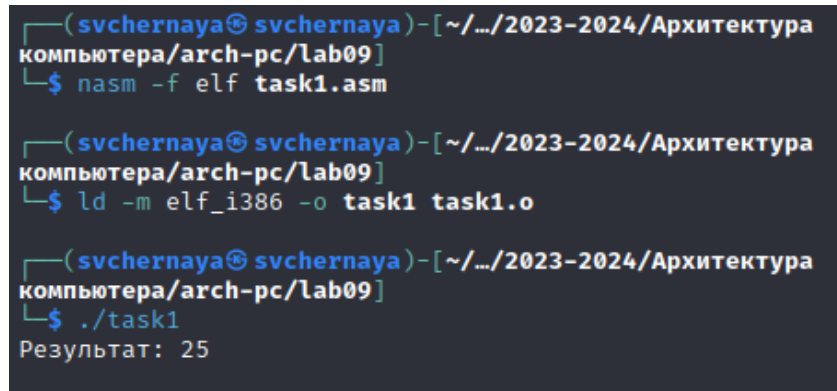
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/task1.asm - Mo
File Edit Search View Document Help
[Icons]
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ——— Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,eax
15 ; ——— Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
21

```

Рис. 4.31: Исправление ошибки

Также, вместо того, чтобы изменять значение `eax`, можно было изменять значение неиспользованного регистра `edx`.

Создаем исполняемый файл и запускаем его. Убеждаемся, что ошибка исправлена. (рис. 4.32)



```
(svchernaya@svchernaya)-[~/.../2023-2024/Архитектура
компьютера/arch-pc/lab09]
└─$ nasm -f elf task1.asm

(svchernaya@svchernaya)-[~/.../2023-2024/Архитектура
компьютера/arch-pc/lab09]
└─$ ld -m elf_i386 -o task1 task1.o

(svchernaya@svchernaya)-[~/.../2023-2024/Архитектура
компьютера/arch-pc/lab09]
└─$ ./task1
Результат: 25
```

Рис. 4.32: Ошибка исправлена

Код программы:

```
%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 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
```

5 Выводы

Во время выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.

6 Список литературы

- [illegible]