

Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Ларина Наталья Денисовна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Реализация подпрограмм в NASM	9
4.2	Отладка программ с помощью GDB	11
4.2.1	Добавление точек останова	15
4.2.2	Работа с данными программы в GDB	16
4.2.3	Обработка аргументов командной строки в GDB	20
4.3	Задания для самостоятельной работы	22
5	Выводы	26
6	Список литературы	27

Список таблиц

Список иллюстраций

4.1	Создание файлов для лабораторной работы	9
4.2	Ввод текста программы из листинга 9.1	10
4.3	Запуск исполняемого файла	10
4.4	Изменение текста программы согласно заданию	11
4.5	Запуск исполняемого файла	11
4.6	Ввод текста программы из листинга 9.2	12
4.7	Получение исполняемого файла	12
4.8	Загрузка исполняемого файла в отладчик	12
4.9	Проверка работы файла с помощью команды run	13
4.10	Установка брейкпоинта и запуск программы	13
4.11	Использование команд disassemble и disassembly-flavor intel	14
4.12	Включение режима псевдографики	15
4.13	Установление точек останова и просмотр информации о них	16
4.14	Использование команды stepi	17
4.15	Просмотр значений переменных	17
4.16	Использование команды set	18
4.17	Вывод значения регистра в разных представлениях и использова- ние команды set для изменения значения регистра	19
4.18	Завершение работы GDB	20
4.19	Копирование файла	20
4.20	Создание файла	20
4.21	Загрузка файла с аргументами в отладчик	21
4.22	Установление точки останова и запуск программы	21
4.23	Просмотр значений, введенных в стек	21
4.24	Написание кода подпрограммы	22
4.25	Запуск программы и проверка его вывода	23
4.26	Ввод текста программы из листинга 9.3	23
4.27	Создание и запуск исполняемого файла	23
4.28	Нахождение причины ошибки	24
4.29	Неверное изменение регистра	24
4.30	Исправление ошибки	25
4.31	Ошибка исправлена	25

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. В общем случае его можно разделить на четыре этапа: • обнаружение ошибки; • поиск её местонахождения; • определение причины ошибки; • исправление ошибки. Можно выделить следующие типы ошибок: • синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • семантические ошибки — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата; • ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль). Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга. Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

GDB (GNU Debugger — отладчик проекта GNU) [1] 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 сторон-них графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки.

4 Выполнение лабораторной работы

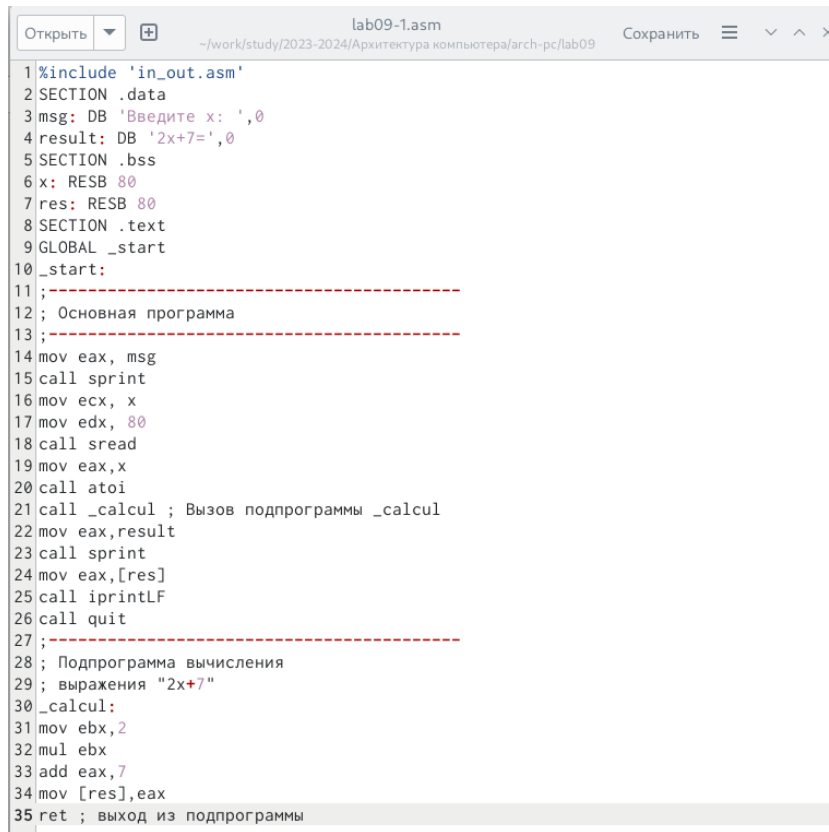
4.1 Реализация подпрограмм в NASM

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

```
ndlarina@dk8n54 ~$ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab09
ndlarina@dk8n54 ~$ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab09
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ touch lab09-1.asm
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ ls
lab09-1.asm
```

Рис. 4.1: Создание файлов для лабораторной работы

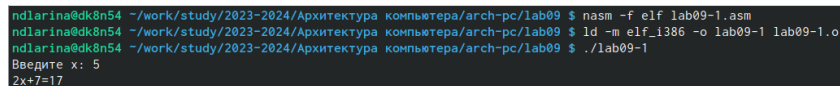
Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис. 4.2)



```
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 ; -----
12 ; Основная программа
13 ; -----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _calcul ; Вызов подпрограммы _calcul
22 mov eax, result
23 call sprint
24 mov eax, [res]
25 call iprintLF
26 call quit
27 ; -----
28 ; Подпрограмма вычисления
29 ; выражения "2x+7"
30 _calcul:
31 mov ebx, 2
32 mul ebx
33 add eax, 7
34 mov [res], eax
35 ret ; выход из подпрограммы
```

Рис. 4.2: Ввод текста программы из листинга 9.1

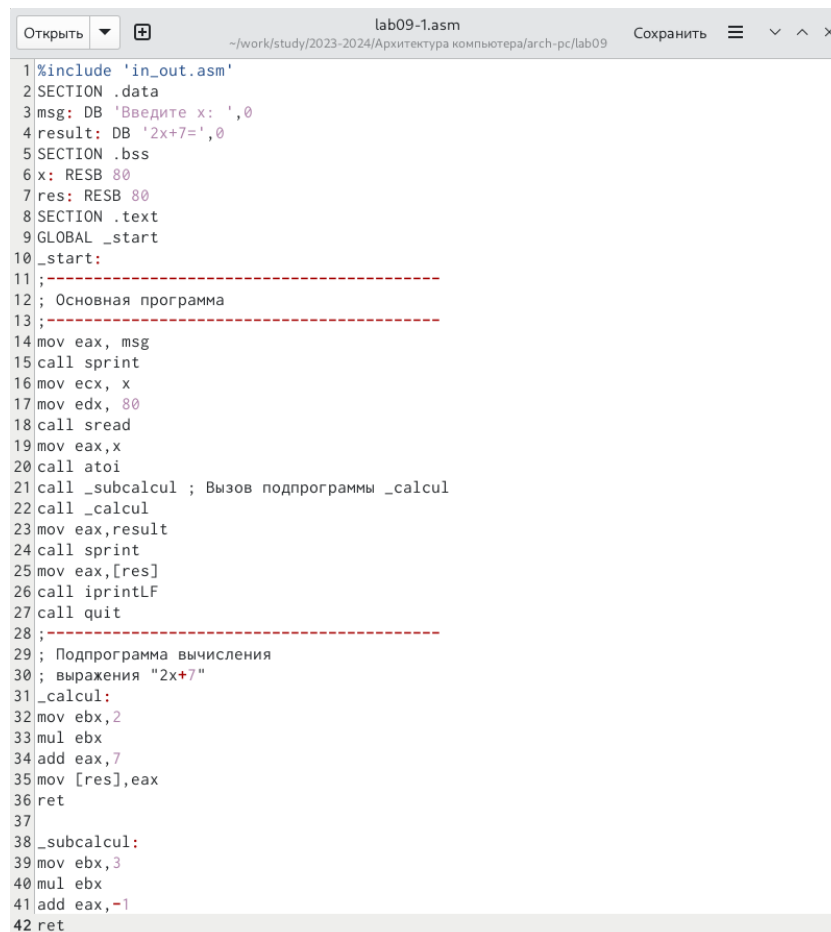
Создаю исполняемый файл и проверяю его работу. (рис. 4.3)



```
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ nasm -f elf lab09-1.asm
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ ./lab09-1
Введите x: 5
2x+7=17
```

Рис. 4.3: Запуск исполняемого файла

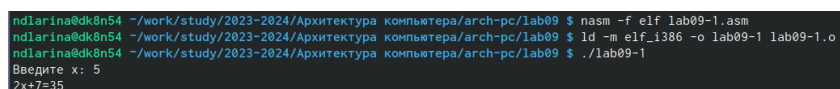
Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul` для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. (рис. 4.4)



```
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 ;-----
12 ; Основная программа
13 ;-----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _subcalcul ; Вызов подпрограммы _calcul
22 call _calcul
23 mov eax, result
24 call sprint
25 mov eax, [res]
26 call iprintLF
27 call quit
28 ;-----
29 ; Подпрограмма вычисления
30 ; выражения "2x+7"
31 _calcul:
32 mov ebx, 2
33 mul ebx
34 add eax, 7
35 mov [res], eax
36 ret
37
38 _subcalcul:
39 mov ebx, 3
40 mul ebx
41 add eax, -1
42 ret
```

Рис. 4.4: Изменение текста программы согласно заданию

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

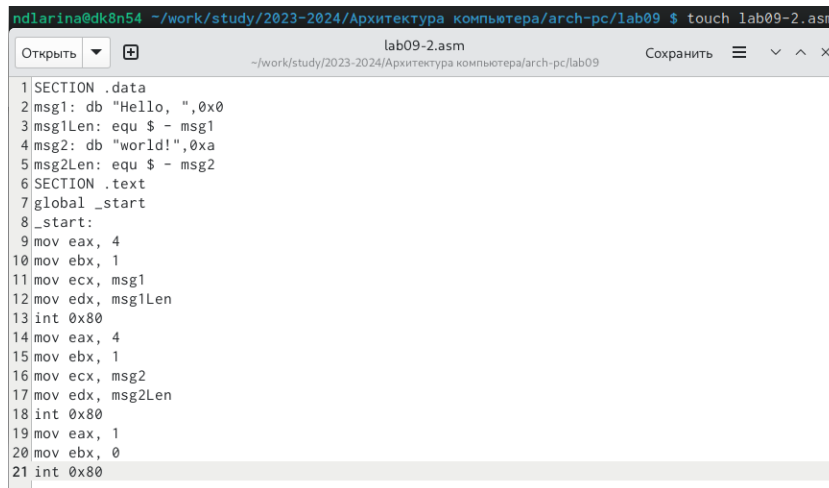


```
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ nasm -f elf lab09-1.asm
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ ./lab09-1
Введите x: 5
2x+7=35
```

Рис. 4.5: Запуск исполняемого файла

4.2 Отладка программ с помощью GDB

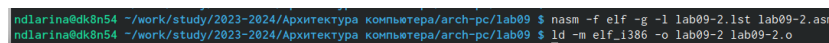
Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (рис. 4.6)



```
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
```

Рис. 4.6: Ввод текста программы из листинга 9.2

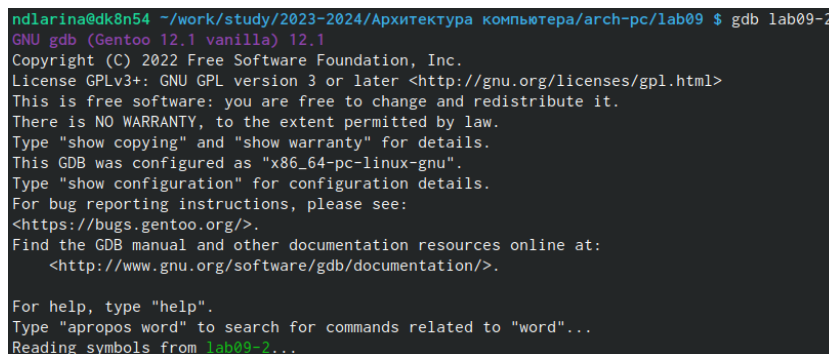
Получаю исполняемый файл для работы с GDB с ключом '-g'. (рис. 4.7)



```
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
```

Рис. 4.7: Получение исполняемого файла

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



```
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 12.1 vanilla) 12.1
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
```

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: /afs/.dk.sci.pfu.edu.ru/home/n/d/ndlarina/work/study/2023-2024/Архитектура_компьютера/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 4852) 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)

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/n/d/ndlarina/work/study/2023-2024/Архитектура_компьютера/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
```

Рис. 4.10: Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный 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.

```

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.

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

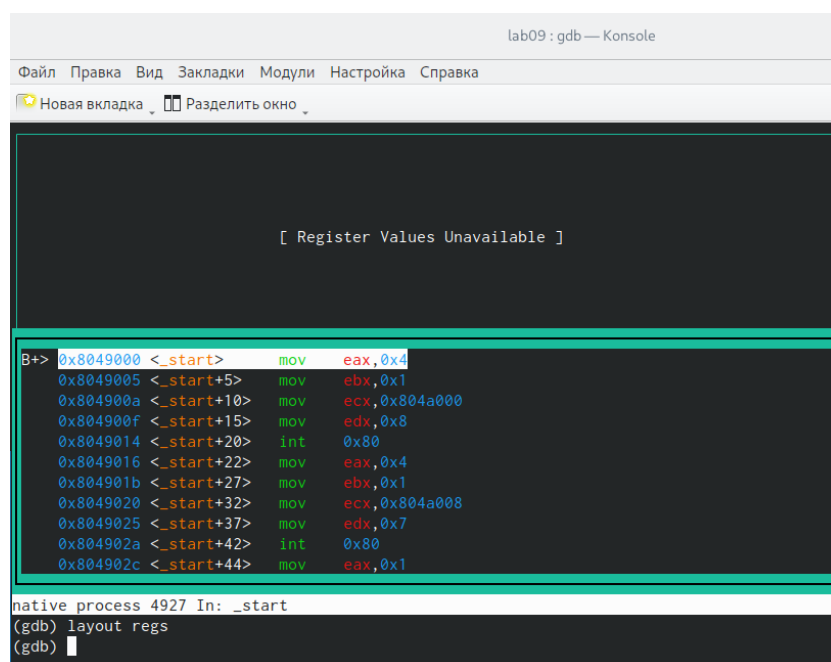


Рис. 4.12: Включение режима псевдографики

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)

The screenshot shows a GDB console window titled 'lab09: gdb — Konsole'. The menu bar includes 'Файл', 'Правка', 'Вид', 'Закладки', 'Модули', 'Настройка', and 'Справка'. Below the menu bar, there are icons for 'Новая вкладка' and 'Разделить окно', and a 'Копировать' button. The main area displays assembly code for a native process 4927. The code starts at address 0x8049000 with the instruction 'mov eax, 0x4'. Subsequent instructions include 'mov ebx, 0x1', 'mov ecx, 0x804a000', 'mov edx, 0x8', 'int 0x80', 'mov eax, 0x4', 'mov ebx, 0x1', 'mov ecx, 0x804a008', 'mov edx, 0x7', 'int 0x80', and 'mov eax, 0x1'. Below the assembly code, the command '(gdb) i b' is entered, showing a list of breakpoints. Breakpoint 1 is at address 0x8049000, file 'lab09-2.asm', line 9, and has been hit 1 time. Breakpoint 2 is at address 0x8049031, file 'lab09-2.asm', line 20. The command '(gdb) b *0x8049031' is entered, and the output shows 'Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.' followed by '(gdb) i b' showing the updated list of breakpoints.

```
lab09: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать

[ Register Values Unavailable ]

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,0x804a008
0x8049025 <_start+37>   mov  edx,0x7
0x804902a <_start+42>   int  0x80
0x804902c <_start+44>   mov  eax,0x1

native process 4927 In: _start
(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint      keep y   0x08049000  lab09-2.asm:9
        breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint      keep y   0x08049000  lab09-2.asm:9
        breakpoint already hit 1 time
2      breakpoint      keep y   0x08049031  lab09-2.asm:20
(gdb)
```

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. 4.1)


```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc3a0 0xffffc3a0

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,0x804a008
0x8049025 <_start+37> mov edx,0x7

native process 3969 In: _start L14 PC: 0x8049016
breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab09-2.asm:20
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si

```

Рис. 4.14: Использование команды stepi

Изменились значения регистров eax, ecx, edx и ebx.

Далее просматриваю значение переменной msg1 по имени с помощью команды x/lb &msg1 и значение переменной msg2 по ее адресу. (рис. 4.15)

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc3a0 0xffffc3a0

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,0x804a008

native process 3969 In: _start L14 PC: 0x8049016
gs      0x0      0
(gdb) x/lb $msg1
Value can't be converted to integer.
(gdb) x/lb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/lb 0x804a008
0x804a008 <msg2>: "world!\n\034"

```

Рис. 4.15: Просмотр значений переменных

С помощью команды set изменяю первый символ переменной msg1 и заменяю

первый символ в переменной msg2. (рис. 4.16)

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc3a0 0xffffc3a0

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,0x804a008

native process 3969 In: _start L14 PC: 0x8049016
0x804a008 <msg2>: "world!\n\034"
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}&msg2='b'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "borld!\n\034"
```

Рис. 4.16: Использование команды set

Затем вывожу в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра edx с помощью команды print r/F \$val. И с помощью команды set изменяю значение регистра ebx в соответствии с заданием. (рис. 4.17)

```
eax      0x4      4
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2

0x8049016 <_start+22> mov     eax,0x4
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,0x804a008
0x8049025 <_start+37> mov     edx,0x7
native process 3969 In: _start L18 PC: 0x804902a
(gdb) sprocess 4587 In: _start L14 PC: 0x8049016
$3 = 1.12103877e-44
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
```

Рис. 4.17: Вывод значения регистра в разных представлениях и 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. 4.18)

```

eax      0x8      8
eax      0x1      1
ecx      0x804a008 134520840
edx      0x7      7

```

```

0x8049016 <_start+22> mov     eax,0x4
0x804900a <_start+10> mov     ecx,0x804a000
0x8049025 <_start+37> mov     edx,0x7
0x804902a <_start+42> int     0x80
> 0x804903a <_start+44> mov     eax,0x1
B+ 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54> int     0x80
0x8049038 add     BYTE PTR [eax],al
native proce... add     BYTE PTR [eax],al
(gdb) sprocess 4587 In: _start
(gdb) p/s $ebx
Breakpoint 2, _start () at lab09-2.asm:20
(gdb) q
A debugging session is active.

Inferior 1 [process 4587] will be killed.

Quit anyway? (y or n)

```

Рис. 4.18: 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 (рис. 4.19)

```
ndlarina@dk8n54 ~ $ cp ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab08/lab8-2.asm ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab09/lab09-3.asm
```

Рис. 4.19: Копирование файла

И затем создаю исполняемый файл. (рис. 4.20)

```
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab09 $
```

Рис. 4.20: Создание файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа `-args`. (рис. 4.21)

```
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент 2 'арг
умент 3'
GNU gdb (Gentoo 12.1 vanilla) 12.1
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb) █
```

Рис. 4.21: Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. 4.22)

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/n/d/ndlarina/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab0
9-3 аргумент1 аргумент 2 аргумент\ 3
Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) █
```

Рис. 4.22: Установление точки останова и запуск программы

Посматриваю вершину стека и позиции стека по их адресам. (рис. 4.23)

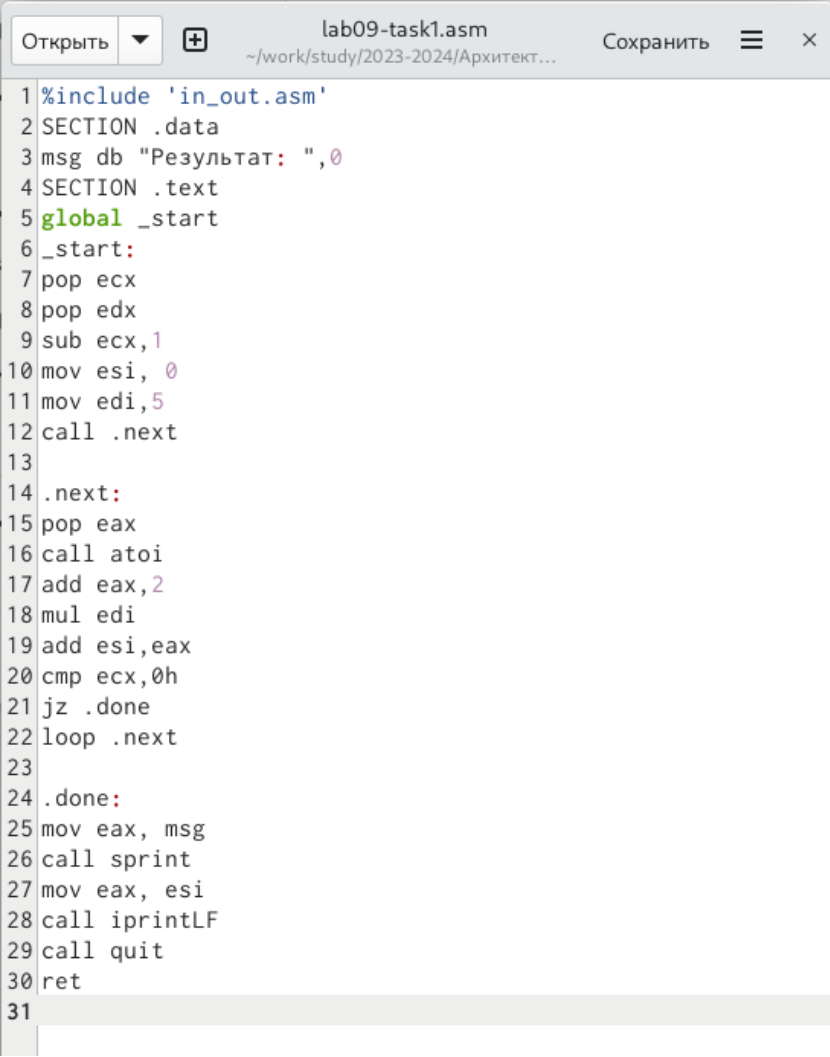
```
(gdb) x/x $esp
0xffffc230: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffc4c5: "/afs/.dk.sci.pfu.edu.ru/home/n/d/ndlarina/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab0
9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffc546: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffc558: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffc569: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffc56b: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb) █
```

Рис. 4.23: 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. 4.24)



```
lab09-task1.asm
~/work/study/2023-2024/Архитект...
Открыть Сохранить

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 mov edi,5
12 call .next
13
14 .next:
15 pop eax
16 call atoi
17 add eax,2
18 mul edi
19 add esi,eax
20 cmp ecx,0h
21 jz .done
22 loop .next
23
24 .done:
25 mov eax, msg
26 call sprint
27 mov eax, esi
28 call iprintLF
29 call quit
30 ret
31
```

Рис. 4.24: Написание кода подпрограммы

Запускаю код и проверяю, что она работает корректно. (рис. 4.25)

```
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ cp ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08/lab8-task1.asm ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab09-task1.asm
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ nasm -f elf lab09-task1.asm
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ ld -m elf_i386 -o lab09-task1 lab09-task1.o
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ ./lab09-task1 1 2 3
Результат: 45
```

Рис. 4.25: Запуск программы и проверка его вывода

2. Ввожу в файл lab09-task2.asm текст программы из листинга 9.3. (рис. 4.26)

```
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
```

Рис. 4.26: Ввод текста программы из листинга 9.3

При корректной работе программы должно выводиться “25”. Создаю исполняемый файл и запускаю его. (рис. 4.27)

```
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ touch lab09-task2.asm
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ nasm -f elf lab09-task2.asm
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ ld -m elf_i386 -o lab09-task2 lab09-task2.o
ndlarina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ ./lab09-task2
Результат: 10
```

Рис. 4.27: 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 esx происходит умножение esx на eax, то есть 4 на 2, вместо умножения 4 на 5 (регистр ebx). Происходит это из-за того, что стоящая перед mov esx,4 инструкция add ebx,eax не связана с mul esx, но связана инструкция mov eax,2. (рис. 4.28)

```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd160 0xffffd160
ebp      0x0      0x0

B+ 0x80490f4 <_start+12> mov    ecx,0x4
B+ 0x80490f6 <_start+17> mul    ecx
B+ 0x80490fb <_start+19> add    ebx,0x5
b+ 0x80490fe <_start+22> mov    edi,ebx
0x8049100 <_start+24> mov    eax,0x804a000

native process 14573 In: _start L13 PC: 0x80490fb
Continuing.

Breakpoint 5, _start () at task2.asm:12
(gdb) c
Continuing.

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

Рис. 4.28: Нахождение причины ошибки

Из-за этого мы получаем неправильный ответ. (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+ 0x80490f6 <_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>

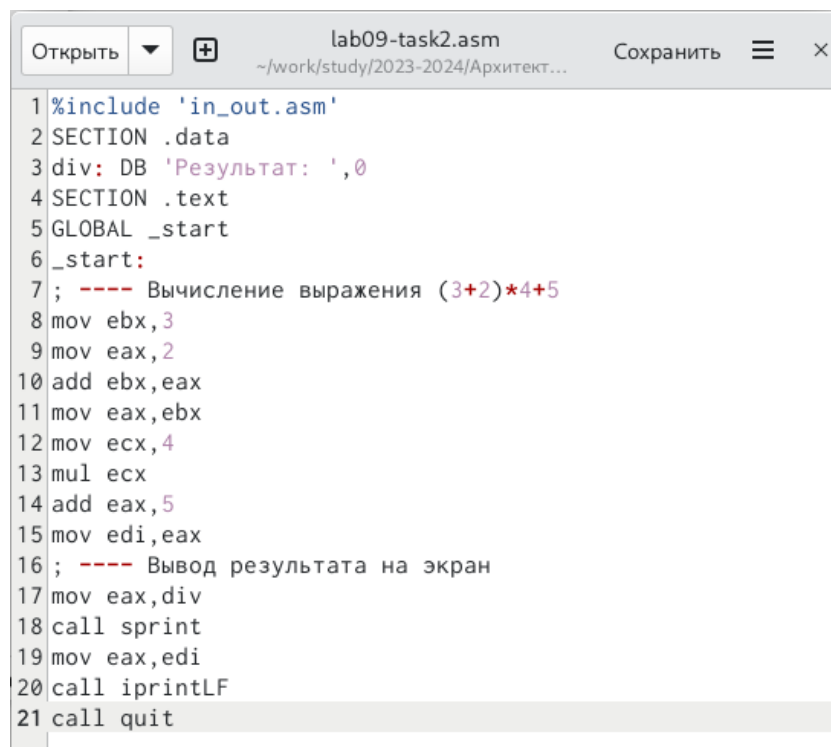
native process 14573 In: _start L14 PC: 0x80490fe
Continuing.

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

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

Рис. 4.29: 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. 4.30)

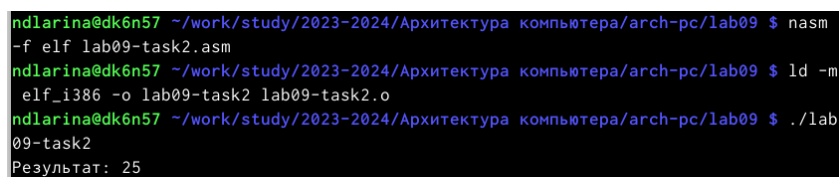


```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov eax,ebx
12 mov ecx,4
13 mul ecx
14 add eax,5
15 mov edi,eax
16 ; ---- Вывод результата на экран
17 mov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

Рис. 4.30: 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. 4.31)



```
ndlarina@dk6n57 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ nasm
-f elf lab09-task2.asm
ndlarina@dk6n57 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ ld -m
elf_i386 -o lab09-task2 lab09-task2.o
ndlarina@dk6n57 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 $ ./lab
09-task2
Результат: 25
```

Рис. 4.31: Ошибка исправлена

5 Выводы

В ходе работы над данной лабораторной работы мне удалось приобрести навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.

6 Список литературы

1. Лабораторная работа №9