

# **Отчёта по лабораторной работе №9**

**Понятие подпрограммы. Отладчик GDB.**

Сидорова Арина Валерьевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>5</b>
2.1	Реализация подпрограмм в NASM . . . . .	5
2.2	Отладка программ с помощью GDB . . . . .	8
<b>3</b>	<b>Выводы</b>	<b>21</b>

# Список иллюстраций

2.1	Создаем каталог с помощью команды <code>mkdir</code> и файл с помощью команды <code>touch</code> . . . . .	5
2.2	Заполняем файл . . . . .	6
2.3	Запускаем файл и проверяем его работу . . . . .	6
2.4	Изменяем файл, добавляя еще одну подпрограмму . . . . .	7
2.5	Запускаем файл и смотрим на его работу . . . . .	7
2.6	Создаем файл . . . . .	8
2.7	Заполняем файл . . . . .	8
2.8	Загружаем исходный файл в отладчик . . . . .	9
2.9	Запускаем программу командой <code>run</code> . . . . .	9
2.10	Запускаем программу с брейкпоинтом . . . . .	9
2.11	Смотрим дисассимилированный код программы . . . . .	10
2.12	Переключаемся на синтаксис Intel . . . . .	10
2.13	Включаем отображение регистров, их значений и результат дисассимилирования программы . . . . .	12
2.14	Используем команду <code>info breakpoints</code> и создаем новую точку останова . . . . .	13
2.15	Смотрим информацию . . . . .	13
2.16	Отслеживаем регистры . . . . .	14
2.17	Смотрим значение переменной . . . . .	14
2.18	Смотрим значение переменной . . . . .	14
2.19	Меняем символ . . . . .	15
2.20	Меняем символ . . . . .	15
2.21	Смотрим значение регистра . . . . .	15
2.22	Изменяем регистр командой <code>set</code> . . . . .	15
2.23	Прописываем команды <code>c</code> и <code>quit</code> . . . . .	16
2.24	Копируем файл . . . . .	16
2.25	Создаем и запускаем в отладчике файл . . . . .	16
2.26	Устанавливаем точку останова . . . . .	16
2.27	Изучаем полученные данные . . . . .	17
2.28	Копируем файл . . . . .	17
2.29	Изменяем файл . . . . .	18
2.30	Проверяем работу программы . . . . .	18
2.31	Создаем файл . . . . .	19
2.32	Изменяем файл . . . . .	19
2.33	Создаем и запускаем файл(работает корректно) . . . . .	20

# 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

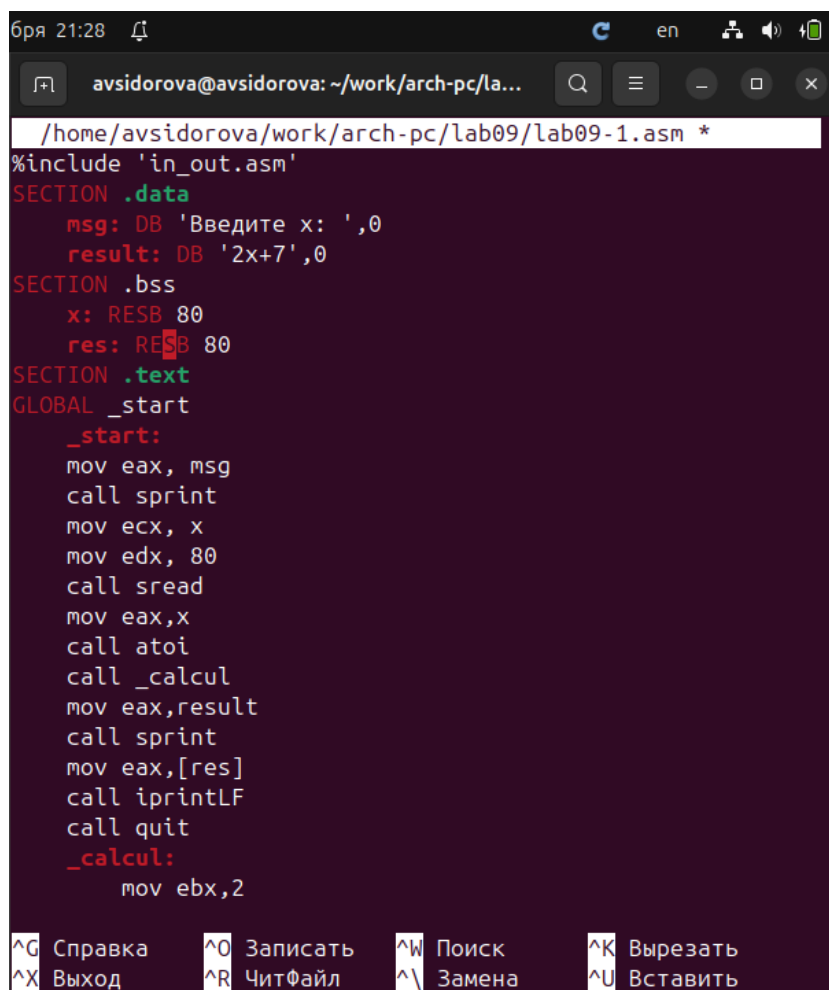
### 2.1 Реализация подпрограмм в NASM

Создаем каталог для программ ЛБ9, и в нем создаем файл (рис. fig. 2.1).

```
avsidorova@avsidorova:~/work/study/2023-2024/Архитектура ко  
avsidorova@avsidorova:~/work/study/2023-2024/Архитектура ко  
мпьютера/arch-pc/labs/lab08$ cd  
avsidorova@avsidorova:~$ mkdir ~/work/arch-pc/lab09  
avsidorova@avsidorova:~$ cd ~/work/arch-pc/lab09  
avsidorova@avsidorova:~/work/arch-pc/lab09$ touch lab09-1.a  
SM  
avsidorova@avsidorova:~/work/arch-pc/lab09$
```

Рис. 2.1: Создаем каталог с помощью команды `mkdir` и файл с помощью команды `touch`

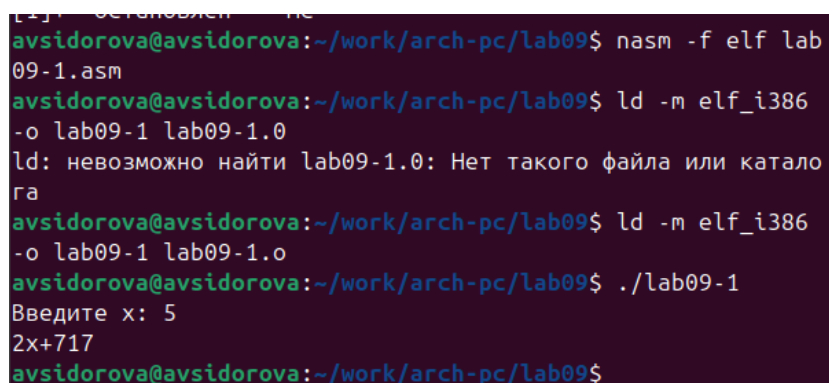
Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.1 (рис. fig. 2.2).



```
бря 21:28 en
avsidorova@avsidorova: ~/work/arch-pc/la...
/home/avsidorova/work/arch-pc/lab09/lab09-1.asm *
#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:
    mov eax, msg
    call sprint
    mov ecx, x
    mov edx, 80
    call sread
    mov eax, x
    call atoi
    call _calcul
    mov eax, result
    call sprint
    mov eax, [res]
    call iprintLF
    call quit
_calcul:
    mov ebx, 2
```

Рис. 2.2: Заполняем файл

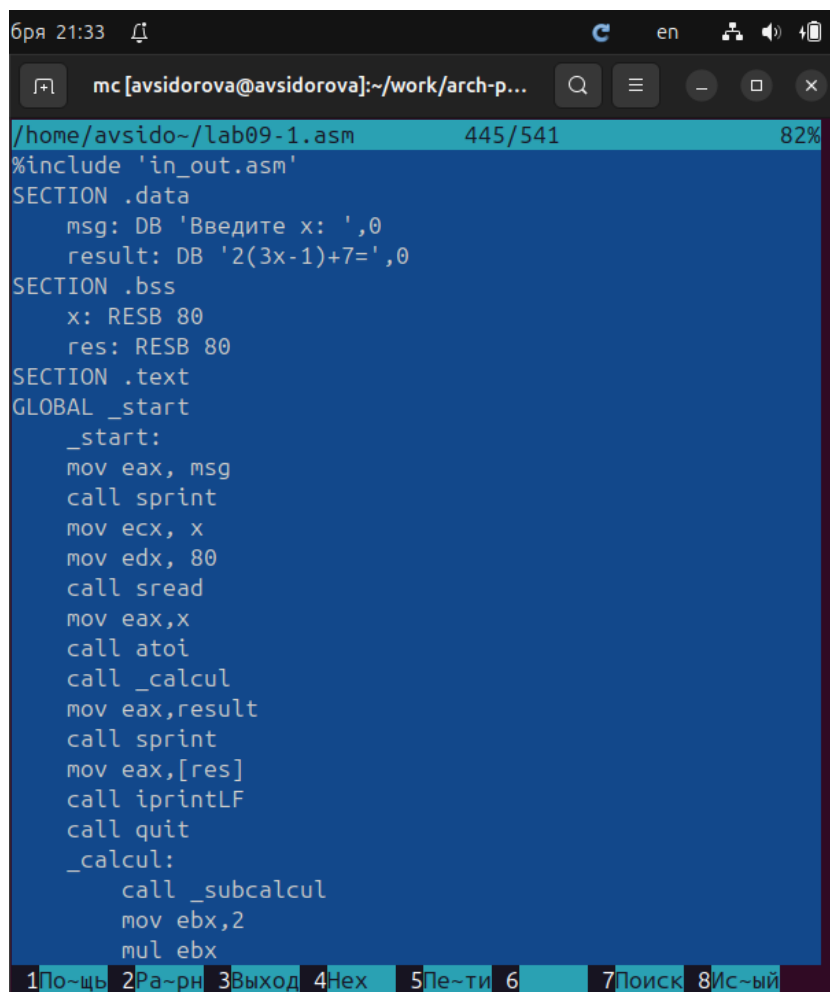
Создаем исполняемый файл и запускаем его (рис. fig. 2.3).



```
avsidorova@avsidorova:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
avsidorova@avsidorova:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
ld: невозможно найти lab09-1.0: Нет такого файла или каталога
avsidorova@avsidorova:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
avsidorova@avsidorova:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+717
avsidorova@avsidorova:~/work/arch-pc/lab09$
```

Рис. 2.3: Запускаем файл и проверяем его работу

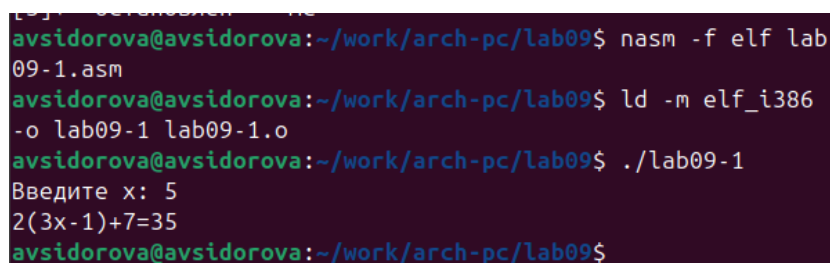
Снова открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму(по условию) (рис. fig. 2.4).



```
бря 21:33
mc [avsidorova@avsidorova]:~/work/arch-p...
/home/avsidorova/lab09-1.asm 445/541 82%
#include 'in_out.asm'
SECTION .data
    msg: DB 'Введите x: ',0
    result: DB '2(3x-1)+7=',0
SECTION .bss
    x: RESB 80
    res: RESB 80
SECTION .text
GLOBAL _start
_start:
    mov eax, msg
    call sprint
    mov ecx, x
    mov edx, 80
    call sread
    mov eax, x
    call atoi
    call _calcul
    mov eax, result
    call sprint
    mov eax, [res]
    call iprintLF
    call quit
_calcul:
    call _subcalcul
    mov ebx, 2
    mul ebx
```

Рис. 2.4: Изменяем файл, добавляя еще одну подпрограмму

Создаем исполняемый файл и запускаем его (рис. fig. 2.5).



```
avsidorova@avsidorova:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
avsidorova@avsidorova:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
avsidorova@avsidorova:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2(3x-1)+7=35
avsidorova@avsidorova:~/work/arch-pc/lab09$
```

Рис. 2.5: Запускаем файл и смотрим на его работу

## 2.2 Отладка программ с помощью GDB

Создаем новый файл в каталоге(рис. fig. 2.6).

```
avsidorova@avsidorova:~/work/arch-pc/lab09$ touch lab09-2.asm  
avsidorova@avsidorova:~/work/arch-pc/lab09$
```

Рис. 2.6: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.2 (рис. fig. 2.7).

```
/home/avsidorova~/lab09-2.asm 363/363 100%  
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. 2.7: Заполняем файл

Получаем исходный файл с использованием отладчика gdb (рис. fig. 2.8).



```
[4]+ Остановлен   мс
avsidorova@avsidorova:~/work/arch-pc/lab09$ nasm -f elf -g
-l lab09-2.lst lab09-2.asm
avsidorova@avsidorova:~/work/arch-pc/lab09$ ld -m elf_i386
-o lab09-2 lab09-2.o
avsidorova@avsidorova:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403
-gdb
Copyright (C) 2024 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".
```

Рис. 2.8: Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. fig. 2.9).

```
(gdb) run
Starting program: /home/avsidorova/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 109625) exited normally]
(gdb)
```

Рис. 2.9: Запускаем программу командой run

Устанавливаем брейкпоинт на метку `_start` и запускаем программу (рис. fig. 2.10).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/avsidorova/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb)
```

Рис. 2.10: Запускаем программу с брейкпоином

Смотрим дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start` (рис. fig. 2.11).

```
(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)
```

Рис. 2.11: Смотрим дисассимилированный код программы

Переключаемся на отображение команд с Intel'овским синтаксисом (рис. fig. 2.12).

```
(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. 2.12: Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel:

1.Порядок операндов: В АТТ синтаксисе порядок операндов обратный, сначала указывается исходный операнд, а затем - результирующий операнд. В Intel синтаксисе порядок обычно прямой, результирующий операнд указывается первым, а исходный - вторым.

2.Разделители: В АТТ синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).

3.Префиксы размера операндов: В АТТ синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как “b” (byte), “w” (word), “l” (long) и “q” (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как “b”, “w”, “d” и “q”.

4.Знак операндов: В АТТ синтаксисе операнды с позитивными значениями предваряются символом “*.Intel*”.

5.Обозначение адресов: В АТТ синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.

6.Обозначение регистров: В АТТ синтаксисе обозначение регистра начинается с символа “%”. В Intel синтаксисе обозначение регистра может начинаться с символа “R” или “E” (например, “%eax” или “RAX”).

Включаем режим псевдографики (рис. fig. 2.13).

```
avsidorova@avsidorova: ~/work/arch-pc/la...
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd030 0xffffd030
ebp      0x0      0x0
esi      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,0x804a008

process 109675 (asm) In: _start      L9      PC: 0x8049000
(gdb) Layout regs
Undefined command: "Layout". Try "help".
(gdb) layout regs
(gdb)
```

Рис. 2.13: Включаем отображение регистров, их значений и результат дисассимилирования программы

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. fig. 2.14).

```

avsidorova@avsidorova: ~/work/arch-pc/la...
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd030 0xffffd030
ebp      0x0      0x0
esi      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,0x804a008

process 109675 (asm) In: _start L9 PC: 0x8049000
(gdb) Layout regs
Undefined command: "Layout". Try "help".
(gdb) layout regs
(gdb) info breakpoints
Num      Type          Disp Enb Address      What
1        breakpoint    keep y 0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb)

```

Рис. 2.14: Используем команду info breakpoints и создаем новую точку останова

Посмотрим информацию о всех установленных точках останова (рис. fig. 2.15).

```

(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. 2.15: Смотрим информацию

Выполняем 5 инструкций командой si (рис. fig. 2.16).

```

register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd030 0xffffd030
ebp      0x0      0x0
esi      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,0x804a008

process 109675 (asm) In: _start      L14      PC: 0x8049016
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) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.16: Отслеживаем регистры

Во время выполнения команд менялись регистры: ebx, ecx, edx, eax, ebp. Смотрим значение переменной msg1 по имени (рис. fig. 2.17).

```

(gdb) si
(gdb) x/lsb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb)

```

Рис. 2.17: Смотрим значение переменной

Смотрим значение переменной msg2 по адресу (рис. fig. 2.18).

```

0x804a000 <msg1>:      Hello,
(gdb) x/lsb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)

```

Рис. 2.18: Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. fig. 2.19).

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

Рис. 2.19: Меняем символ

Изменим первый символ переменной msg2 (рис. fig. 2.20).

```
(gdb) set {char}&msg2 = 'L'
(gdb) x/lb &msg2
0x804a008 <msg2>:      "Lorld!\n\034"
(gdb)
```

Рис. 2.20: Меняем символ

Смотрим значение регистра edx в разных форматах (рис. fig. 2.21).

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

Рис. 2.21: Смотрим значение регистра

Изменяем регистр ebx (рис. fig. 2.22).

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

Рис. 2.22: Изменяем регистр командой set

Выводится разные значения, так как команда без кавычек присваивает регистру вводимое значение.

Прописываем команды для завершения программы и выхода из GDB (рис. fig. 2.23).

```

(gdb) p/$5
$5 = 2
(gdb) c
Continuing.
World!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb)

```

Рис. 2.23: Прописываем команды c и quit

Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. fig. 2.24).

```

avsidorova@avsidorova:~/work/arch-pc/lab09$ cp ~/work/arch-
pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
avsidorova@avsidorova:~/work/arch-pc/lab09$

```

Рис. 2.24: Копируем файл

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. fig. 2.25).

```

avsidorova@avsidorova:~/work/arch-pc/lab09$ nasm -f elf -g
-l lab09-3.lst lab09-3.asm
avsidorova@avsidorova:~/work/arch-pc/lab09$ ld -m elf_i386
-o lab09-3 lab09-3.o
avsidorova@avsidorova:~/work/arch-pc/lab09$ gdb --args lab0
9-3 2 3 '5'

```

Рис. 2.25: Создаем и запускаем в отладчике файл

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. fig. 2.26).

```

Reading symbols from lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/avsidorova/work/arch-pc/lab09/lab09
-3 2 3 5

```

Рис. 2.26: Устанавливаем точку останова

Смотрим позиции стека по разным адресам (рис. fig. 2.27).



```

(gdb) x/x $esp
0xffffd020: 0x00000004
(gdb) x/s *(void**)( $esp +4)
0xffffd1e0: "/home/avsidorova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp +8)
0xffffd20c: "2"
(gdb) x/s *(void**)( $esp +12)
0xffffd20e: "3"
(gdb) x/s *(void**)( $esp +16)
0xffffd210: "5"
(gdb) x/s *(void**)( $esp +20)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 2.27: Изучаем полученные данные

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита(4 байта).

##Задание для самостоятельной работы

Задание 1

Копируем файл lab8-4.asm(ср №1 в ЛБ8) в файл с именем lab09-3.asm (рис. fig. 2.28).

```

[6]+ Остановлен   gdb --args lab09-3 2 3 5
avsidorova@avsidorova:~/work/arch-pc/lab09$ cp ~/work/arch-
pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
avsidorova@avsidorova:~/work/arch-pc/lab09$ mc

[7]+ Остановлен   mc
avsidorova@avsidorova:~/work/arch-pc/lab09$ nasm -f elf lab
09-5.asm

```

Рис. 2.28: Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму (рис. fig. 2.29).

```
/home/avsidorova/work/arch-pc/lab09/lab09-4.asm *
x: RESB 80
res: RESB 90
SECTION .text
global _start
_start:
    mov eax, msg
    call sprint
    mov ecx, x
    mov edx, 80
    call sread
    mov eax, x
    call atoi
    call _calcul
    mov eax, result
    call sprint
    mov eax, [res]
    call iprintLF
    call quit
_calcul:
    add eax, 10
    mov ebx, 3
    mul ebx
    mov [res], eax
    ret
```

Имя файла для записи: </lab09-4.asm

^G Справка	M-D Формат DOS	M-A Доп. в нач	M-B Резерв. копия
^C Отмена	M-M Формат Mac	M-P Доп. в кон	^T Обзор

Рис. 2.29: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. fig. 2.30).

```
avsidorova@avsidorova:~/work/arch-pc/lab09$ ./lab09-4
Введите x: 7
3(10+x)=51
avsidorova@avsidorova:~/work/arch-pc/lab09$
```

Рис. 2.30: Проверяем работу программы

## Задание 2

Создаем новый файл в дирректории (рис. fig. 2.31).

```

avsidorova@avsidorova:~/work/arch-pc/lab09$ touch lab09-5.asm
avsidorova@avsidorova:~/work/arch-pc/lab09$ mc

[8]+  Остановлен   mc
avsidorova@avsidorova:~/work/arch-pc/lab09$

```

Рис. 2.31: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. fig. 2.32).

```

/home/avsidorova/work/arch-pc/lab09/lab09-5.asm *
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
    mov eax,3
    mov ebx,2
    add eax,ebx
    mov ecx,4
    mul ecx
    add eax,5
    mov edi,eax
    mov eax,div
    call sprint
    mov eax,edi
    call iprintLF
    call quit

```

<sup>^</sup>G Справка    <sup>^</sup>O Записать    <sup>^</sup>W Поиск    <sup>^</sup>K Вырезать  
<sup>^</sup>X Выход    <sup>^</sup>R ЧитФайл    <sup>^</sup>\ Замена    <sup>^</sup>U Вставить

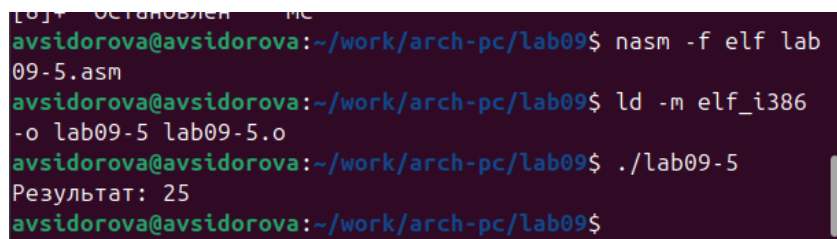
Рис. 2.32: Изменяем файл

Создаем исполняемый файл и запускаем его (работает неправильно)

Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на изменение регистров командой si

Изменяем программу для корректной работы

Создаем исполняемый файл и запускаем его (рис. fig. 2.33).

A terminal window with a dark purple background and green text. The prompt is 'avsidorova@avsidorova:~/work/arch-pc/lab09\$'. The first command is 'nasm -f elf lab09-5.asm', which produces no output. The second command is 'ld -m elf\_i386 -o lab09-5 lab09-5.o', which also produces no output. The third command is './lab09-5', which outputs 'Результат: 25'. The prompt returns to 'avsidorova@avsidorova:~/work/arch-pc/lab09\$'.

```
avsidorova@avsidorova:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
avsidorova@avsidorova:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
avsidorova@avsidorova:~/work/arch-pc/lab09$ ./lab09-5
Результат: 25
avsidorova@avsidorova:~/work/arch-pc/lab09$
```

Рис. 2.33: Создаем и запускаем файл(работает корректно)

## 3 Выводы

Приобрели навыки написания программ с использованием подпрограмм. Познакомились с методами отладки при помощи GDB и его основными возможностями.