

# **Отчёт по лабораторной работе №9**

**дисциплина: Архитектура компьютера**

Люкшина Влада Алексеевна

# Содержание

<b>1)Цель работы</b>	<b>6</b>
<b>2)Задание</b>	<b>7</b>
<b>3)Выполнение лабораторной работы</b>	<b>8</b>
3.1) Создаем каталог для выполнения лабораторной работы № 9, переходим в него и создаем файл lab09-1.asm. . . . .	8
3.2) Вводим в файл lab09-1.asm текст программы из листинга 9.1. Создаем исполняемый файл и проверяем его работу. . . . .	8
3.3) Изменяем текст программы, добавив подпрограмму _subcalcul в подпрограмму _calcul. . . . .	9
3.4) Проверяем работу файла. . . . .	9
3.5) Создаем файл lab09-2.asm с текстом программы из Листинга 9.2. .	10
3.6) Получаем исполняемый файл. Для работы с GDB в исполняемый файл добавляем отладочную информацию, используя ключ -g. . .	10
3.7) Загружаем исполняемый файл в отладчик gdb. . . . .	11
3.8) Проверяем работу программы, запустив ее в оболочке GDB с помощью команды run. . . . .	11
3.9) Устанавливаем брейкпоинт на метку _start и запускаем программу. .	12
3.10) Просматриваем дисассемблированный код программы с помощью команды disassemble начиная с метки _start. . . . .	12
3.11) Переключаемся на отображение команд с Intel'овским синтаксисом с помощью команды set disassembly-flavor intel. . . . .	13
3.12) Включаем режим псевдографики для более удобного анализа программы. . . . .	14
3.13) Проверяем, установлена ли точка останова по имени метки(_start). .	15
3.14) Определяем адрес предпоследней инструкции (mov ebx,0x0) и устанавливаем точку останова. . . . .	16
3.15) Смотрим информацию о всех установленных точках останова. . .	17
3.16) Выполняем 5 инструкций с помощью команды stepi (или si). . . .	18
3.17) Смотрим значение переменной msg1 по имени. . . . .	19
3.18) Смотрим значение переменной msg2 по адресу. . . . .	19
3.19) Изменяем первый символ переменной msg1. . . . .	19
3.20) Изменяем символ во второй переменной msg2. . . . .	19
3.21) Выводим в различных форматах значение регистра edx. . . . .	20
3.22) С помощью команды set изменяем значение регистра ebx. . . . .	20

3.23) Завершаем выполнение программы с помощью команды <code>continue</code> (сокращенно <code>c</code> ) или <code>stepi</code> (сокращенно <code>si</code> ) и выходим из GDB с помощью команды <code>quit</code> (сокращенно <code>q</code> ). . . . .	21
3.24) Копируем файл <code>lab8-2.asm</code> в файл с именем <code>lab09-3.asm</code> . . . . .	21
3.25) Создаем исполняемый файл. . . . .	22
3.26) Загружаем исполняемый файл в отладчик, указав аргументы. . .	22
3.27) Устанавливаем точку останова перед первой инструкцией в программе и запускаем ее. . . . .	23
3.28) Просматриваем позиции стека по разным адресам. . . . .	23
<b>4) Самостоятельная работа</b>	<b>24</b>
4.1) Копируем файл <code>lab8-3.asm</code> в файл <code>lab09-4.asm</code> . . . . .	24
4.2) Изменяем файл, реализуя вычисления значения через подпрограмму(вариант 3, 10х-5) . . . . .	25
4.3) Создаем файл <code>lab09-5.asm</code> и записываем туда текст из листинга. . .	26
4.4) Создаем исполняемый файл и проверяем его работу. . . . .	26
4.5) Создаем исполняемый файл, запускаем его в отладчике GDB и ищем ошибку. . . . .	27
4.6) Изменяем программу. . . . .	28
4.7) Создаем исполняемый файл и проверяем его работу. . . . .	28
<b>5) Выводы</b>	<b>29</b>

# Список иллюстраций

1	Создаем каталог, переходим в него и создаем файл . . . . .	8
2	Вводим текст, создаем файл и проверяем его работу . . . . .	8
3	Изменяем текст программы . . . . .	9
4	Проверяем . . . . .	9
5	Создаем файл и вводим текст из листинга . . . . .	10
6	Получаем исполняемый файл . . . . .	10
7	Загружаем файл . . . . .	11
8	Проверяем . . . . .	11
9	Устанавливаем брейкпоинт и запускаем программу . . . . .	12
10	Просматриваем код . . . . .	12
11	Переключаемся и смотрим программу . . . . .	13
12	Включаем режим псевдографики . . . . .	14
13	Проверяем . . . . .	15
14	Устанавливаем точку останова . . . . .	16
15	Смотрим . . . . .	17
16	Смотрим значение переменной . . . . .	19
17	Смотрим значение переменной . . . . .	19
18	Изменяем символ . . . . .	19
19	Изменяем символ . . . . .	19
20	Выводим значение . . . . .	20
21	Завершаем выполнение программы . . . . .	21
22	Копируем файл . . . . .	21
23	Создаем исполняемый файл . . . . .	22
24	Создаем исполняемый файл . . . . .	22
25	Устанавливаем точку и запускаем . . . . .	23
1	Копируем . . . . .	24
2	Изменяем файл . . . . .	25
3	Создаем файл . . . . .	26
4	Проверяем. Программа работает неправильно . . . . .	26
5	Проверяем регистры . . . . .	27
6	Изменяем программу . . . . .	28
7	Проверяем . . . . .	28

## **Список таблиц**

# **1)Цель работы**

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

## **2)Задание**

Написать программы с использованием вызова подпрограммы, проанализировать их. Преобразовать программу из лабораторной работы №8.

## 3)Выполнение лабораторной работы

**3.1) Создаем каталог для выполнения лабораторной работы № 9, переходим в него и создаем файл lab09-1.asm.**

```
lyukshinava@fedora:~$ mkdir ~/work/arch-pc/lab09
lyukshinava@fedora:~$ cd ~/work/arch-pc/lab09
lyukshinava@fedora:~/work/arch-pc/lab09$ touch lab09-1.asm
lyukshinava@fedora:~/work/arch-pc/lab09$
```

Рис. 1: Создаем каталог, переходим в него и создаем файл

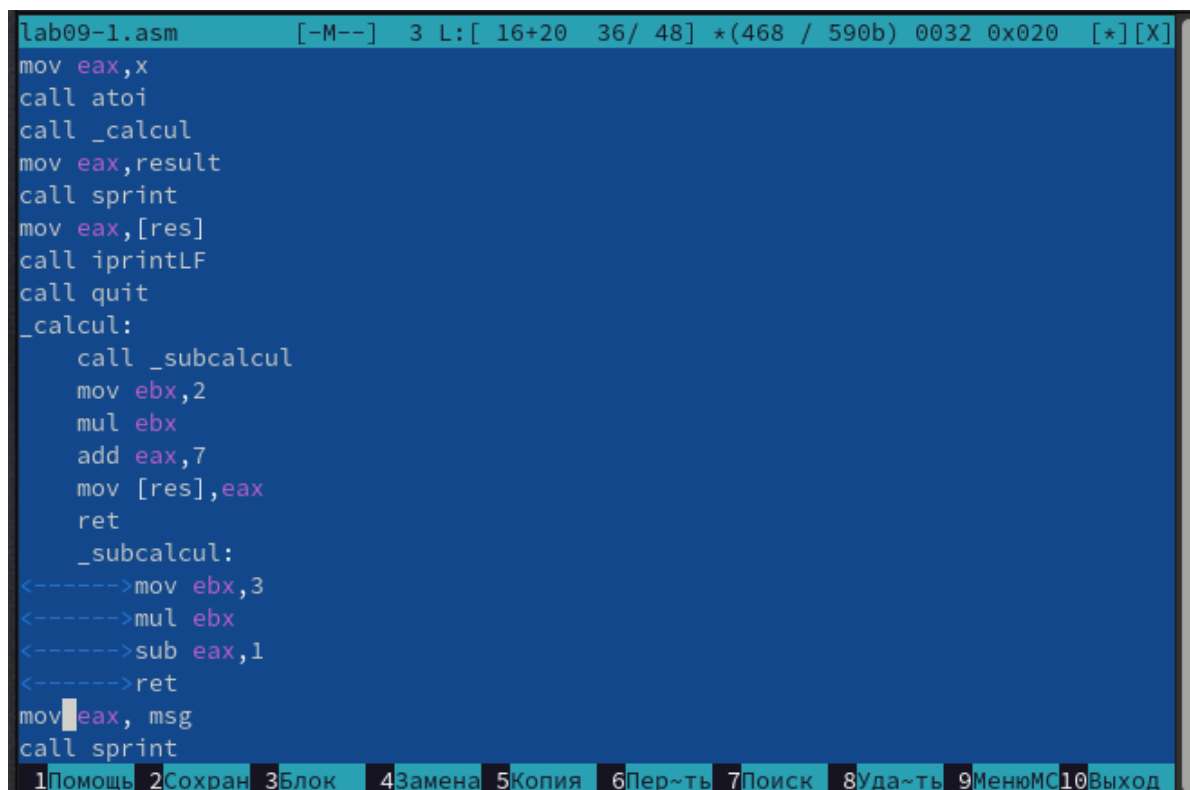
**3.2) Вводим в файл lab09-1.asm текст программы из листинга 9.1. Создаем исполняемый файл и проверяем его работу.**

```
lyukshinava@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
lyukshinava@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
lyukshinava@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 2
2x+7=11
lyukshinava@fedora:~/work/arch-pc/lab09$
```

Рис. 2: Вводим текст, создаем файл и проверяем его работу



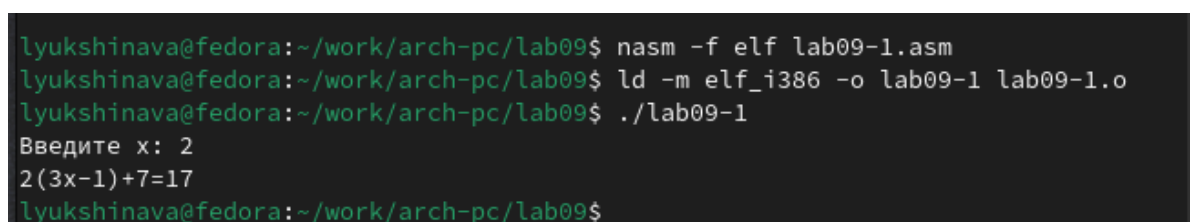
### 3.3) Изменяем текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`.



```
lab09-1.asm [-M--] 3 L:[ 16+20 36/ 48] *(468 / 590b) 0032 0x020 [*][X]
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
    add eax,7
    mov [res],eax
    ret
    _subcalcul:
<----->mov ebx,3
<----->mul ebx
<----->sub eax,1
<----->ret
mov eax, msg
call sprint
```

Рис. 3: Изменяем текст программы

### 3.4) Проверяем работу файла.



```
lyukshinava@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
lyukshinava@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
lyukshinava@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 2
2(3x-1)+7=17
lyukshinava@fedora:~/work/arch-pc/lab09$
```

Рис. 4: Проверяем

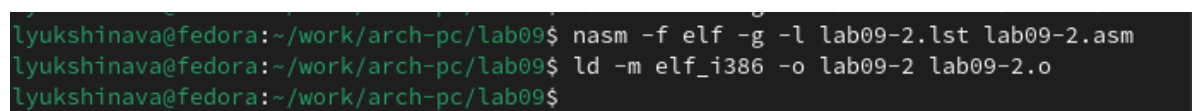
### 3.5) Создаем файл lab09-2.asm с текстом программы из Листинга 9.2.



```
lab09-2.asm  [----]  0 L:[ 1+ 0  1/ 22] *(0  / 294b) 0083 0x053  [*] [X]
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
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Уда~ть 9МенюМС10Выход
```

Рис. 5: Создаем файл и вводим текст из листинга

### 3.6) Получаем исполняемый файл. Для работы с GDB в исполняемый файл добавляем отладочную информацию, используя ключ -g.



```
lyukshinava@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
lyukshinava@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
lyukshinava@fedora:~/work/arch-pc/lab09$
```

Рис. 6: Получаем исполняемый файл

### 3.7) Загружаем исполняемый файл в отладчик gdb.

```
lyukshinava@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 15.2-1.fc40
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-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 lab09-2...
(gdb) █
```

Рис. 7: Загружаем файл

### 3.8) Проверяем работу программы, запустив ее в оболочке GDB с помощью команды run.

```
(gdb) run
Starting program: /home/lyukshinava/work/arch-pc/lab09/lab09-2

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.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 5824) exited normally]
(gdb) █
```

Рис. 8: Проверяем

### 3.9) Устанавливаем брейкпоинт на метку `_start` и запускаем программу.

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

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

Рис. 9: Устанавливаем брейкпоинт и запускаем программу

### 3.10) Просматриваем дисассемблированный код программы с помощью команды `disassemble` начиная с метки `_start`.

```
(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. 10: Просматриваем код

### 3.11) Переключаемся на отображение команд с Intel'овским синтаксисом с помощью команды `set disassembly-flavor intel`.

```
(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. 11: Переключаемся и смотрим программу

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel:

- 1) В АТТ синтаксисе обозначение регистра начинается с символа “%”. В Intel синтаксисе обозначение регистра может начинаться с символа “R” или “E”.
- 2) 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.
- 3) В АТТ синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.
- 4) В АТТ синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как “b” (byte). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как “b”, “w”, “d” и “q”.
- 5) В АТТ синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).

### 3.12) Включаем режим псевдографики для более удобного анализа программы.

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

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
0x8049031 <_start+49> mov    ebx,0x0

native process 6035 (asm) In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb)
```

Рис. 12: Включаем режим псевдографики

### 3.13) Проверяем, установлена ли точка останова по имени метки(\_start).

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

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
0x8049031 <_start+49> mov    ebx,0x0

native process 6035 (asm) In: _start L9 PC: 0x8049000
(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) □
```

Рис. 13: Проверяем

### 3.14) Определяем адрес предпоследней инструкции (mov ebx,0x0) и устанавливаем точку останова.

```
Register group: general
esp      0xffffd050      0xffffd050
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

0x8049836  add     BYTE PTR [eax],al
0x8049838  add     BYTE PTR [eax],al
0x804983a  add     BYTE PTR [eax],al
0x804983c  add     BYTE PTR [eax],al
0x804983e  add     BYTE PTR [eax],al
0x8049840  add     BYTE PTR [eax],al
0x8049842  add     BYTE PTR [eax],al
0x8049844  add     BYTE PTR [eax],al
0x8049846  add     BYTE PTR [eax],al
0x8049848  add     BYTE PTR [eax],al
0x804984a  add     BYTE PTR [eax],al
0x804984c  add     BYTE PTR [eax],al

native process 6035 (asm) In: _start L9 PC: 0x8049000
(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. 14: Устанавливаем точку останова



### 3.15) Смотрим информацию о всех установленных точках останова.

```
(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. 15: Смотрим

### 3.16) Выполняем 5 инструкций с помощью команды stepi (или si).

```
Register group: general
esp      0xffffd050      0xffffd050
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

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
b+ 0x8049031 <_start+49>  mov     ebx,0x0

native process 6035 (asm) In: _start L14 PC: 0x8049016
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) 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) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Во время выполнения команд изменились регистры ebx, ecx, edx, eax, eip.

### 3.17) Смотрим значение переменной msg1 по имени.

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

Рис. 16: Смотрим значение переменной

### 3.18) Смотрим значение переменной msg2 по адресу.

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

Рис. 17: Смотрим значение переменной

### 3.19) Изменяем первый символ переменной msg1.

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

Рис. 18: Изменяем символ

### 3.20) Изменяем символ во второй переменной msg2.

```
(gdb) set {char}&msg2='M'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "Morld!\n\034"
(gdb)
```

Рис. 19: Изменяем символ

### 3.21) Выводим в различных форматах значение регистра edx.

```
(gdb) p/t $edx
$4 = 1000
(gdb) p/s $edx
$5 = 8
(gdb) p/x $edx
$6 = 0x8
(gdb)
```

Рис. 20: Выводим значение

### 3.22) С помощью команды set изменяем значение регистра ebx.

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$7 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$8 = 2
(gdb)
```

Команда без кавычек присваивает регистру вводимое значение, поэтому при выведении значения регистра мы получаем разные результаты.

**3.23) Завершаем выполнение программы с помощью команды `continue` (сокращенно `c`) или `stepi` (сокращенно `si`) и выходим из GDB с помощью команды `quit` (сокращенно `q`).**

```
(gdb) c
Continuing.
World!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb) quit
A debugging session is active.

    Inferior 1 [process 6035] will be killed.

Quit anyway? (y or n)
```

Рис. 21: Завершаем выполнение программы

**3.24) Копируем файл `lab8-2.asm` в файл с именем `lab09-3.asm`.**

```
(gdb) copy-to-asm
lyukshinava@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
lyukshinava@fedora:~/work/arch-pc/lab09$
```

Рис. 22: Копируем файл

### 3.25) Создаем исполняемый файл.

```
lyukshinava@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
lyukshinava@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
lyukshinava@fedora:~/work/arch-pc/lab09$
```

Рис. 23: Создаем исполняемый файл

### 3.26) Загружаем исполняемый файл в отладчик, указав аргументы.

```
lyukshinava@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 2 3 '5'
GNU gdb (Fedora Linux) 15.2-1.fc40
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-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 lab09-3...
(gdb)
```

Рис. 24: Создаем исполняемый файл

### 3.27) Устанавливаем точку останова перед первой инструкцией в программе и запускаем ее.

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 7.
(gdb) run
Starting program: /home/lyukshinava/work/arch-pc/lab09/lab09-3 2 3 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 lab09-3.asm:7
7      pop ecx
(gdb) █
```

Рис. 25: Устанавливаем точку и запускаем

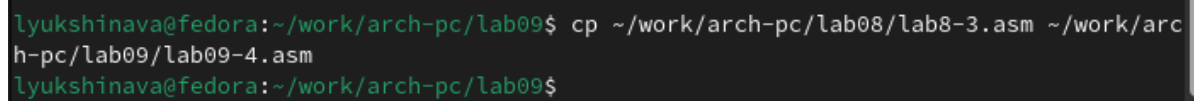
### 3.28) Просматриваем позиции стека по разным адресам.

```
(gdb) x/s *(void**)(esp + 4)
0xffffd201:    "/home/lyukshinava/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd22e:    "2"
(gdb) x/s *(void**)(esp + 12)
0xffffd230:    "3"
(gdb) x/s *(void**)(esp + 16)
0xffffd232:    "5"
(gdb) x/s *(void**)(esp + 20)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)
```

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 4 байта.

## 4) Самостоятельная работа

### 4.1) Копируем файл lab8-3.asm в файл lab09-4.asm

A terminal window with a dark background and green text. It shows a user named lyukshinava@fedora in the directory ~/work/arch-pc/lab09. The user enters the command 'cp ~/work/arch-pc/lab08/lab8-3.asm ~/work/arch-pc/lab09/lab09-4.asm'. The command is executed successfully, and the prompt returns.

```
lyukshinava@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-3.asm ~/work/arch-pc/lab09/lab09-4.asm
lyukshinava@fedora:~/work/arch-pc/lab09$
```

Рис. 1: Копируем



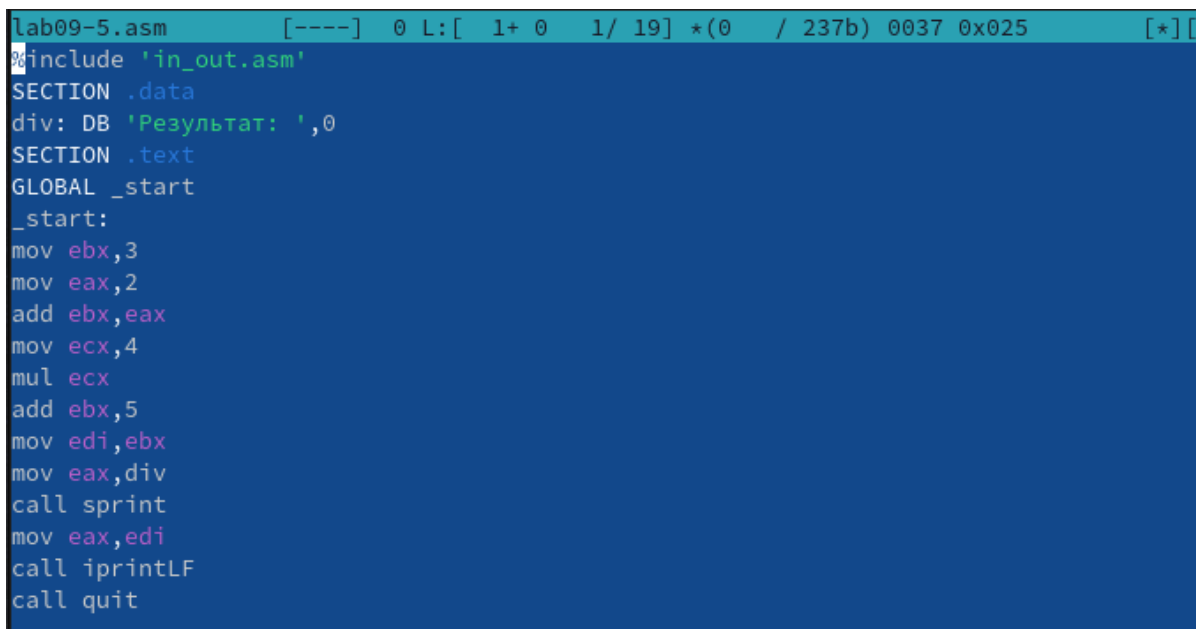
## 4.2) Изменяем файл, реализуя вычисления значения через подпрограмму(вариант 3, 10х-5)

```
%include 'in_out.asm'
SECTION .data
    msg db "Результат: ", 0
SECTION .bss
    b resb 80
SECTION .text
    global _start

_start:
    pop ecx
    cmp ecx, 0
    jle _end
    mov dword [b], 0
    mov esi, 10
next:
    pop eax
    call atoi
    imul eax, esi
    sub eax, 5
    add [b], eax
    loop next
_end:
    mov eax, msg
    call sprint
    mov eax, dword [b]
    call iprintLF
    call quit
```

Рис. 2: Изменяем файл

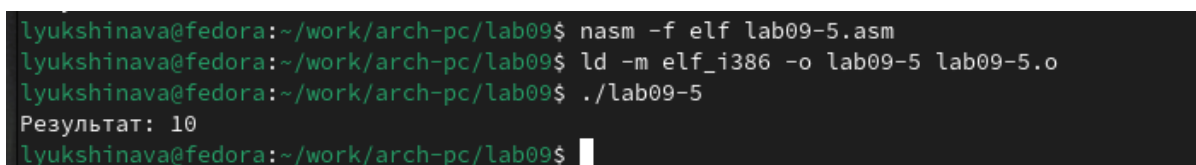
### 4.3) Создаем файл lab09-5.asm и записываем туда текст из листинга.



```
lab09-5.asm [----] 0 L: [ 1+ 0 1/ 19] *(0 / 237b) 0037 0x025 [*] [
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 3: Создаем файл

### 4.4) Создаем исполняемый файл и проверяем его работу.



```
lyukshinava@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
lyukshinava@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
lyukshinava@fedora:~/work/arch-pc/lab09$ ./lab09-5
Результат: 10
lyukshinava@fedora:~/work/arch-pc/lab09$
```

Рис. 4: Проверяем. Программа работает неправильно

## 4.5) Создаем исполняемый файл, запускаем его в отладчике GDB и ищем ошибку.

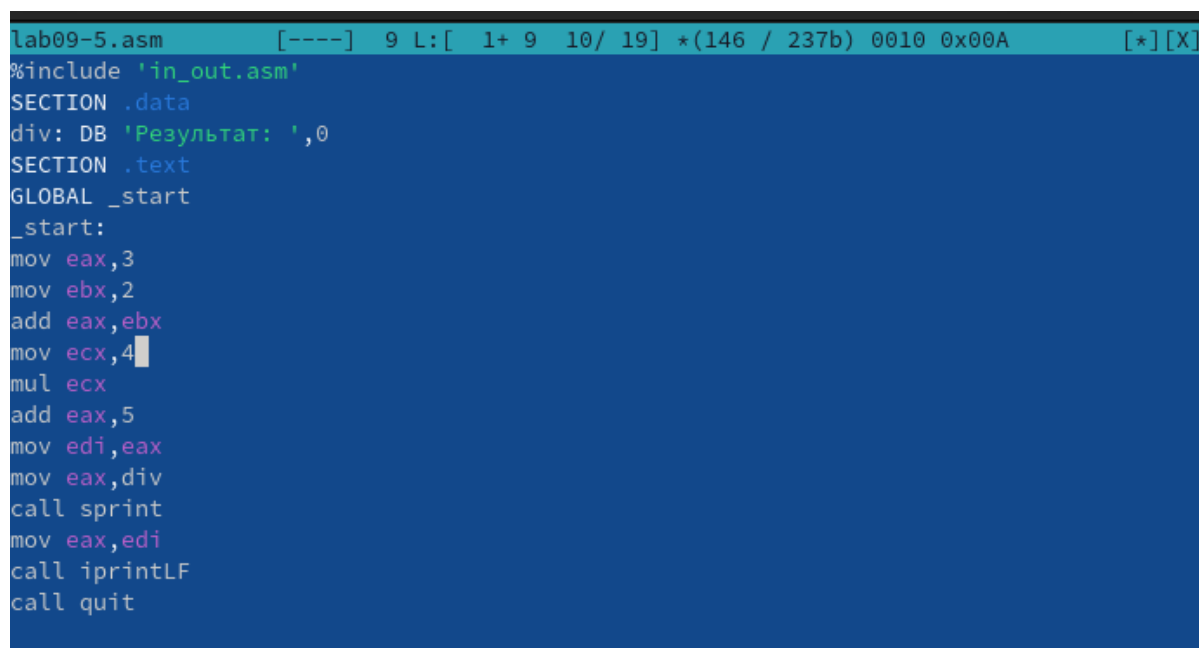
```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490ed 0x80490ed <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43

B+ 0x80490e8 <_start>      mov     ebx,0x3
>0x80490ed <_start+5>     mov     eax,0x2
0x80490f2 <_start+10>     add     ebx,eax
0x80490f4 <_start+12>     mov     ecx,0x4
0x80490f9 <_start+17>     mul     ecx
0x80490fb <_start+19>     add     ebx,0x5
0x80490fe <_start+22>     mov     edi,ebx
0x8049100 <_start+24>     mov     eax,0x804a000
0x8049105 <_start+29>     call    0x804900f <sprint>
0x804910a <_start+34>     mov     eax,edi
0x804910c <_start+36>     call    0x8049086 <iprintLF>
0x8049111 <_start+41>     call    0x80490db <quit>
0x8049116      add     BYTE PTR [eax],al
0x8049118      add     BYTE PTR [eax],al

native process 13999 (asm) In: _start      L8      PC: 0x80490ed
(gdb) layout regs
(gdb) si
(gdb) █
```

Рис. 5: Проверяем регистры

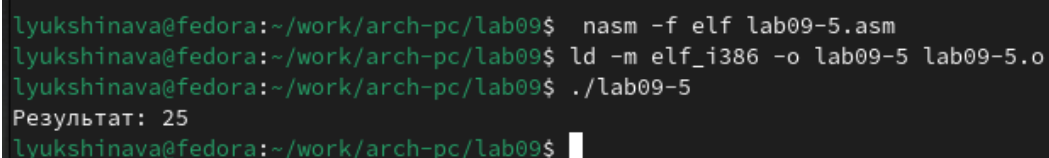
## 4.6) Изменяем программу.



```
lab09-5.asm [----] 9 L: [ 1+ 9 10/ 19] *(146 / 237b) 0010 0x00A [*] [X]
#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
```

Рис. 6: Изменяем программу

## 4.7) Создаем исполняемый файл и проверяем его работу.



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

Рис. 7: Проверяем

## **5)Выводы**

Мы познакомились с методами отладки при помощи GDB и его возможностями.