

# **Отчёт по лабораторной работе №9**

**Дисциплина: архитектура компьютера**

Маваси Башар

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>5</b>
<b>3</b>	<b>Самостоятельная работа</b>	<b>17</b>
3.1	Здание №1 . . . . .	17
3.2	Задание №2 . . . . .	18
<b>4</b>	<b>Вывод</b>	<b>23</b>

# Список иллюстраций

2.1	Создание директории . . . . .	5
2.2	Редактирование файла . . . . .	6
2.3	Запуск исполняемого файла . . . . .	6
2.4	Редоктирование . . . . .	7
2.5	Запуск исполняемого файла . . . . .	7
2.6	Создание исполняемого файла . . . . .	9
2.7	Работа с отладчиком . . . . .	9
2.8	Дисассамблирование кода . . . . .	10
2.9	Синтаксис Intel . . . . .	10
2.10	Режим псевдографики . . . . .	11
2.11	Просмотр точек останова . . . . .	12
2.12	Вывод значений регистров . . . . .	13
2.13	Вывод значений переменных . . . . .	13
2.14	Изменение значений переменных . . . . .	14
2.15	Изменение значений переменных . . . . .	14
2.16	Запуск исполняемого файла . . . . .	14
2.17	Вывод значений переменных . . . . .	14
2.18	Вывод значений переменных . . . . .	15
2.19	Запуск отладчика . . . . .	15
2.20	Изменение значений переменных . . . . .	16
2.21	Просмотр содержимого в esp . . . . .	16
2.22	Вывод значений переменных . . . . .	16
3.1	Изменение программы . . . . .	17
3.2	Запуск исполняемого файла . . . . .	18
3.3	Программа вычисления выражения $(3 + 2) * 4 + 5$ . . . . .	18
3.4	Запуск файла в отладчике . . . . .	19
3.5	Запуск программы . . . . .	19
3.6	Работа с отладчиком . . . . .	20
3.7	Проверка значений регистров . . . . .	21
3.8	Выявление главных ошибок . . . . .	21
3.9	Исправление ошибок в программе . . . . .	22
3.10	Запуск исполняемого файла . . . . .	22

# 1 Цель работы

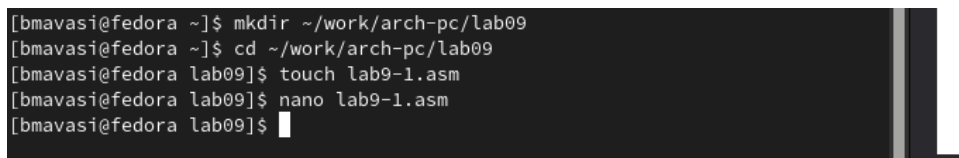
Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

---

## 2 Выполнение лабораторной работы

### Шаг 1

С помощью утилиты `mkdir` создаю директорию `lab09`, перехожу в нее и создаю файл для работы. (рис. [2.1])

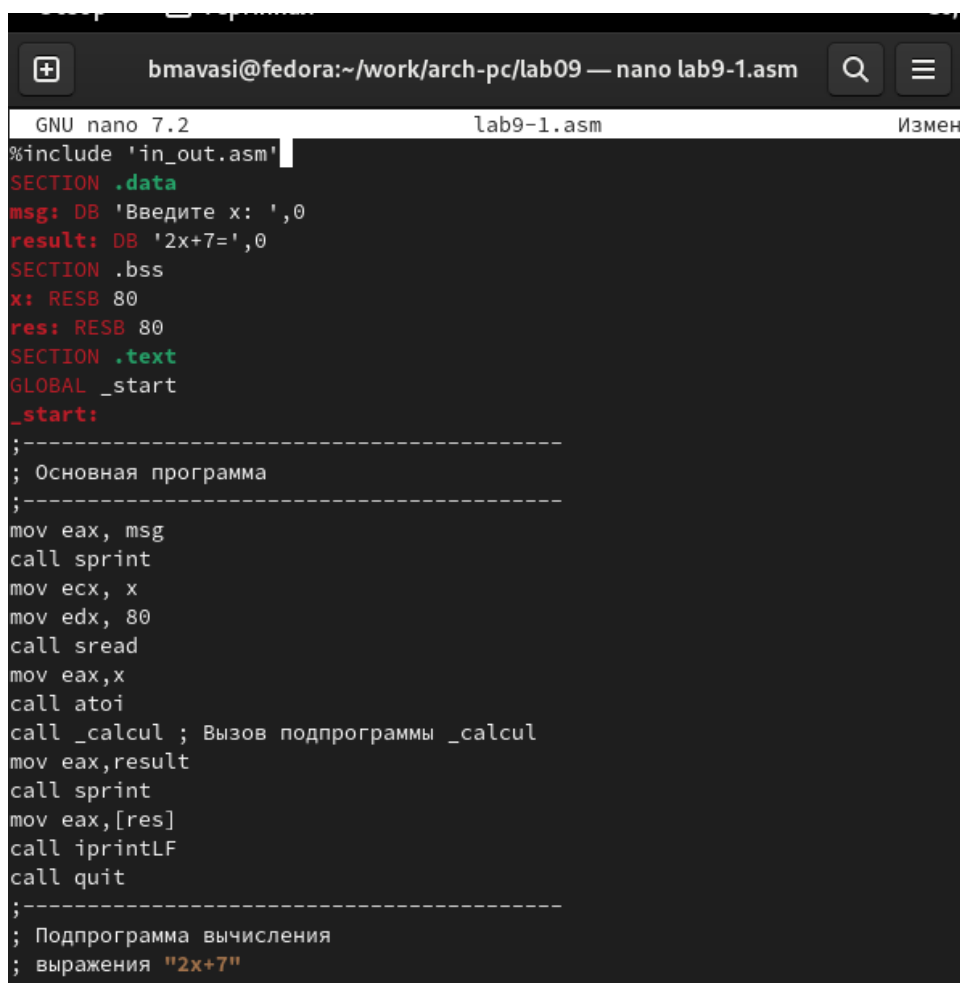


```
[bmavasi@fedora ~]$ mkdir ~/work/arch-pc/lab09
[bmavasi@fedora ~]$ cd ~/work/arch-pc/lab09
[bmavasi@fedora lab09]$ touch lab9-1.asm
[bmavasi@fedora lab09]$ nano lab9-1.asm
[bmavasi@fedora lab09]$
```

Рис. 2.1: Создание директории

### Шаг 2

Открываю созданный файл `lab9-1.asm`, вставляю в него программу с использованием подпрограммы(рис.[2.2]).

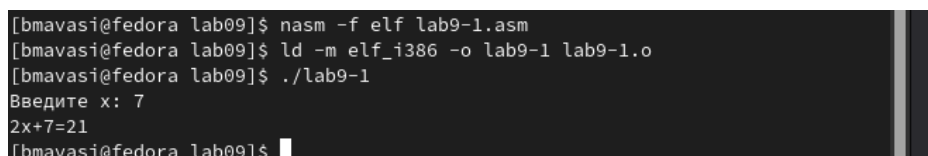


```
GNU nano 7.2 lab9-1.asm
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
```

Рис. 2.2: Редактирование файла

### Шаг 3

Создаю исполняемый файл программы и запускаю его (рис. [2.3]).



```
[bmavasi@fedora lab09]$ nasm -f elf lab9-1.asm
[bmavasi@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[bmavasi@fedora lab09]$ ./lab9-1
Введите x: 7
2x+7=21
[bmavasi@fedora lab09]$
```

Рис. 2.3: Запуск исполняемого файла

### Шаг 4

Изменяю текст программы для вычисления композиции  $f$  от  $g$ , при  $g(x) = 3x-1$ .

Создаю новую подпрограмму `_subcalcul` для вычисления функции  $g$  (рис. [2.4]).

```
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы

_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
mov [res], eax

ret
```

Рис. 2.4: Редоктирование

## Шаг 5

Создаю исполняемый файл и проверяю работу программы (рис. [2.5]).

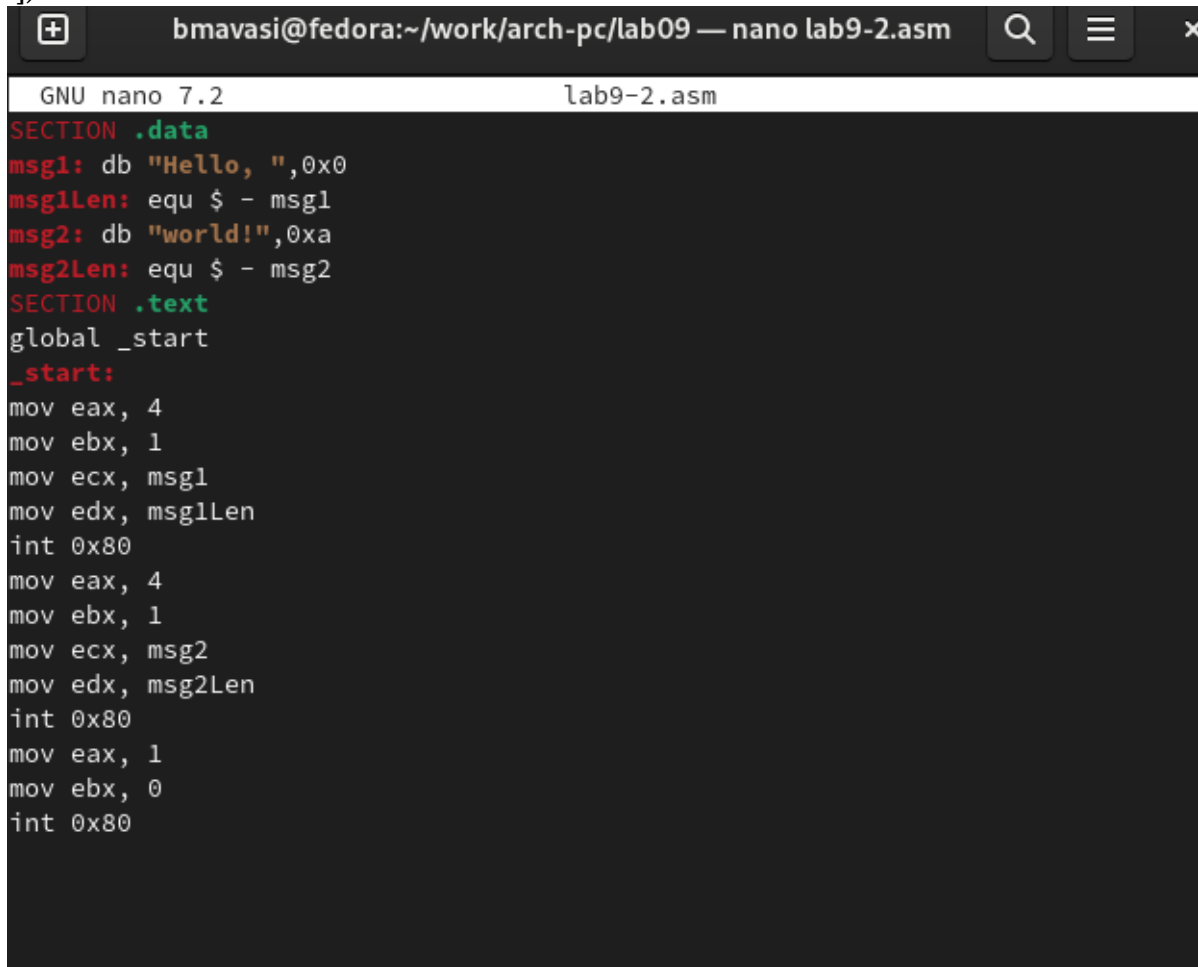
```
[bmavasi@fedora lab09]$ nasm -f elf lab9-1.asm
[bmavasi@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[bmavasi@fedora lab09]$ ./lab9-1
Введите x: 6
2x+7=41
[bmavasi@fedora lab09]$
```

Рис. 2.5: Запуск исполняемого файла

- Программа отработала верно!!

## Шаг 6

Создаю новый файл lab9-2.asm и вставляю в него текст из Листинга 9.2 (рис. [??]).

A screenshot of a terminal window with the nano text editor. The title bar shows the user 'bmavasi' at 'fedora' in the directory '~/work/arch-pc/lab09', editing 'lab9-2.asm'. The editor content shows assembly code for two data strings, 'Hello, ' and 'world!', followed by a text section with two print instructions. The code is as follows:

```
GNU nano 7.2 lab9-2.asm
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
```

{ #fig:006 width=80% }

### Шаг 7

Создаю исполняемый файл, файл листинга для работы с отладчиком GDB (рис. [2.6]).



```
[bmavasi@fedora lab09]$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
[bmavasi@fedora lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[bmavasi@fedora lab09]$ ./lab9-2
Hello, world!
[bmavasi@fedora lab09]$ gdb lab9-2
GNU gdb (GDB) Fedora Linux 13.2-6.fc38
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-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 lab9-2...
(gdb)
```

Рис. 2.6: Создание исполняемого файла

## Шаг 8

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды `run`, и для более подробного анализа программы, вставляю брэйкпоинт на метку `_start` (рис. [2.7]).

```
(gdb) run
Starting program: /home/bmavasi/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 11184) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) █
```

Рис. 2.7: Работа с отладчиком

## Шаг 9

Посмотрим дизассемблированный код, начиная с этой метки. (рис. [2.8]).

```

(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. 2.8: Дисассамблирование кода

## Шаг 10

Так же посмотрим как выглядит дизассемблированный код с синтаксисом Intel (рис. [2.9]).

```

(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. 2.9: Синтаксис Intel

- В представлении АТТ в виде 16-ричного числа записаны первые аргументы всех команд, а в представлении intel так записываются адреса вторых

аргументов.

## Шаг 11

Включим режим псевдографики, с помощью которого отображается код программы и содержимое регистров (рис. [2.10]).

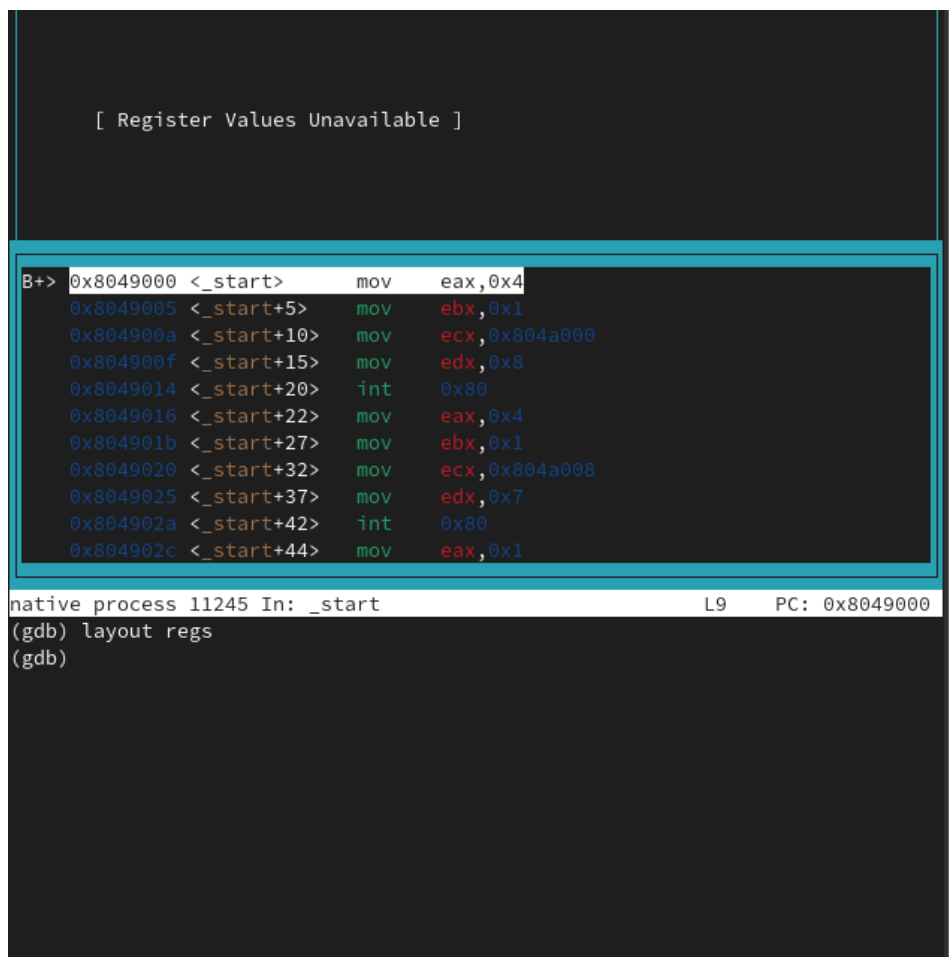


Рис. 2.10: Режим псевдографики

## Шаг 12

Посмотрим информацию о наших точках останова и сразу добавим еще одну точку.(рис. [2.11]).

```

[ Register Values Unavailable ]

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
0x8049036 <_start+54>  int     0x80

exec No process In:                               L??  PC: ??
(gdb) layout asm
(gdb) layout regs
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab9-2.asm:9
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab9-2.asm:9
2        breakpoint keep y  0x08049031 lab9-2.asm:20
(gdb)

```

Рис. 2.11: Просмотр точек останова

## Шаг 13

Так же можно выводить значения регистров. Делается это командой `i r`. Псевдографика предствалена на (рис. [2.12]).

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]

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 11411 In: _start L9 PC: 0x8049000
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 2.12: Вывод значений регистров

## Шаг 14

В отладчике можно вывести текущее значение переменных. Сделать это можно по имени или по адресу: выводим значения переменных msg1 и msg2 (рис. [2.13]).

```
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb) 
```

Рис. 2.13: Вывод значений переменных

## Шаг 15

Так же отладчик позволяет менять значения переменных прямо во время выполнения программы (рис. [2.14]).

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x604a000 <msg1>:      "hello, "
(gdb) █
```

Рис. 2.14: Изменение значений переменных

- Заменяю первый символ 'H' на 'h'

## Шаг 16

Заменяю первый символ переменной msg2 на символ j. (рис. [2.15]).

```
(gdb) set {char}&msg2='k'
(gdb) x/1sb &msg2
0x604a008 <msg2>:      "korld!\n\034"
(gdb) █
```

Рис. 2.15: Изменение значений переменных

## Шаг 17

Выводить можно так же содержимое регистров. Выведем значение ebx в разных форматах. . (рис. [2.16]).

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

Рис. 2.16: Запуск исполняемого файла

## Шаг 18

Как и переменным, регистрам можно задавать значения (рис. [2.17]).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) p/t $ebx
$2 = 110010
(gdb) p/x $ebx
$3 = 0x32
(gdb) █
```

Рис. 2.17: Вывод значений переменных

## Шаг 19

Так же отладчик позволяет менять значения переменных прямо во время выполнения программы (рис. [2.18]).

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

Рис. 2.18: Вывод значений переменных

- Однако при попытке задать строчное значение, происходит ошибка.

**Завершим работу в gdb командами `continue`, она закончит выполнение программы, и `exit`, она завершит сеанс gdb**

## Шаг 20

Скопируем файл из лабораторной 9, переименуем её и создадим исполняемый файл. Откроем отладчик и зададим аргументы. (рис. [2.19]).

```
[bmavasi@fedora lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[bmavasi@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[bmavasi@fedora lab09]$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora Linux 13.2-6.fc38
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-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. 2.19: Запуск отладчика

## Шаг 21

Создадим точку останова на метке `_start` и запустим программу(рис. [2.20]).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/darfonos/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) █
```

Рис. 2.20: Изменение значений переменных

## Шаг 22

Посмотрим на содержимое стека, что расположено по адресу, находящемуся в регистре esp(рис. [2.21]).

```
(gdb) x/x $esp
0xffffd160: 0x00000005
(gdb) █
```

Рис. 2.21: Просмотр содержимого в esp

## Шаг 23

Далее посмотрим на все остальные аргументы в стеке. Их адреса располагаются в 4 байтах друг от друга(именно столько занимает элемент стека) (рис. [2.22]).

```
(gdb) x/s *(void**)($esp + 4)
0xffffd31b: "/home/darfonos/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp)
0x5: <error: Cannot access memory at address 0x5>
(gdb) x/s *(void**)($esp + 4)
0xffffd31b: "/home/darfonos/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd345: "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd357: "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd368: "2"
(gdb) x/s *(void**)($esp + 20)
A syntax error in expression, near `'.
(gdb) █
```

Рис. 2.22: Вывод значений переменных

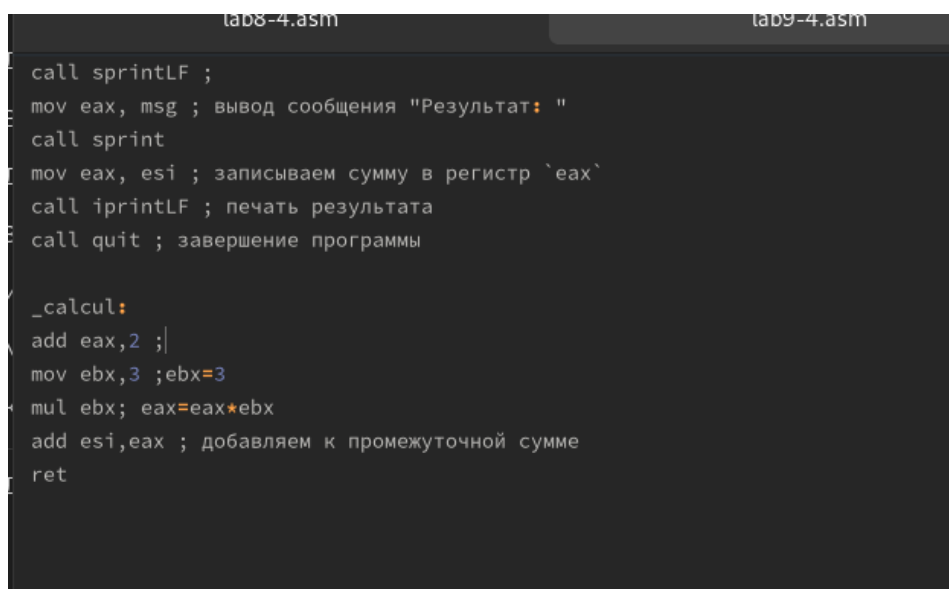


## 3 Самостоятельная работа

### 3.1 Задание №1

#### Шаг 1

Копирую программу из лабораторной 8 и переименовываю его. Изменяю текст программы с использованием подпрограммы (рис. [3.1]).



```
lab8-4.asm      lab9-4.asm
call sprintLF ;
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы

_calcul:
add eax,2 ;|
mov ebx,3 ;ebx=3
mul ebx; eax=eax*ebx
add esi,eax ; добавляем к промежуточной сумме
ret
```

Рис. 3.1: Изменение программы

#### Шаг 3

Создаю исполняемый файл и проверяю работу изменённой программы .(рис. [3.2]).

```

[bmavasi@fedora lab09]$ nasm -f elf -g -l lab9-4.lst lab9-4.asm
[bmavasi@fedora lab09]$ ld -m elf_i386 -o lab9-4 lab9-4.o
[bmavasi@fedora lab09]$ ./lab9-4 1 2 3 4
Функция: f(x)=3(x+2)
Результат: 54
[bmavasi@fedora lab09]$

```

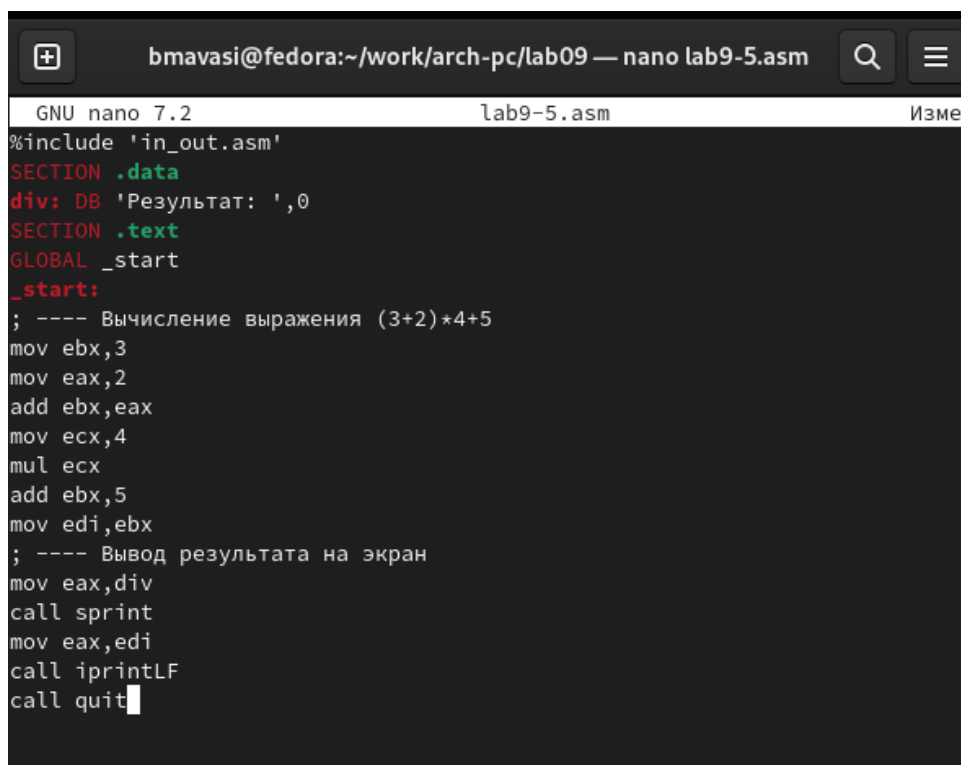
Рис. 3.2: Запуск исполняемого файла

**Программа отработала верно**

## 3.2 Задание №2

### Шаг 1

Создаю новый файл и вставляю в него программу из листинга (рис. [3.3]).



```

GNU nano 7.2                                lab9-5.asm                                Изме
%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 ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 3.3: Программа вычисления выражения  $(3 + 2) * 4 + 5$

### Шаг 2

Запускаю программу в отладчике и проверяю его работу и вижу, что результат вычисления неправильный. (рис. [3.5]).

```
[bmavasi@fedora lab09]$ nasm -f elf -g -l lab9-5.lst lab9-5.asm
[bmavasi@fedora lab09]$ ld -m elf_i386 -o lab9-5 lab9-5.o
[bmavasi@fedora lab09]$ gdb lab9-5
GNU gdb (GDB) Fedora Linux 13.2-6.fc38
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-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 lab9-5...
(gdb) █
```

Рис. 3.4: Запуск файла в отладчике

```
(gdb) run
Starting program: /home/darfonos/work/arch-pc/lab09/lab9-5

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.
Результат: 10
```

Рис. 3.5: Запуск программы

### Шаг 3

Для того, чтобы найти ошибку дисассемблирую программу и добавляю брейк-поинты в основной части программы (рис. [3.6]).

```

0x080490e8 <+0>:    mov     ebx,0x3
0x080490ed <+5>:    mov     eax,0x2
0x080490f2 <+10>:   add     ebx,eax
0x080490f4 <+12>:   mov     ecx,0x4
0x080490f9 <+17>:   mul     ecx
0x080490fb <+19>:   add     ebx,0x5
0x080490fe <+22>:   mov     edi,ebx
0x08049100 <+24>:   mov     eax,0x804a000
0x08049105 <+29>:   call   0x804900f <sprint>
0x0804910a <+34>:   mov     eax,edi
0x0804910c <+36>:   call   0x8049086 <iprintLF>
0x08049111 <+41>:   call   0x80490db <quit>
End of assembler dump.
(gdb) b *0x080490f4
Breakpoint 1 at 0x80490f4: file lab9-5.asm, line 11.
(gdb) b *0x08049100
Breakpoint 2 at 0x8049100: file lab9-5.asm, line 16.
(gdb)

```

Рис. 3.6: Работа с отладчиком

## Шаг 4

Запускаю программу до первой точки останова, и проверяю значения регистров.

- Замечаю, что результат сложения записывается в регистр ebx. (рис. [3.7]).

```

Register group: general
eax      0x2          2          ecx      0x0          0
edx      0x0          0          ebx      0x5          5
esp      0xffffd1b0   0xffffd1b0   ebp      0x0          0x0
esi      0x0          0          edi      0x0          0
eip      0x80490f4     0x80490f4 <_start+12> eflags   0x206        [ PF IF ]
cs       0x23         35          ss       0x2b         43
ds       0x2b         43          es       0x2b         43
fs       0x0          0          gs       0x0          0

0x80490e8 <_start>      mov     ebx,0x3
0x80490ed <_start+5>    mov     eax,0x2
0x80490f2 <_start+10>   add     ebx,eax
B-> 0x80490f4 <_start+12> mov     ecx,0x4
0x80490f9 <_start+17>   mul     ecx,ecx
0x80490fb <_start+19>   add     ebx,0x5
0x80490fe <_start+22>   mov     edi,ebx
b+ 0x8049100 <_start+24> mov     eax,0x804a000
0x8049105 <_start+29>   call    0x804900f <sprint>
0x804910a <_start+34>   mov     eax,edi
0x804910c <_start+36>   call    0x8049086 <iprintf>
0x8049111 <_start+41>   call    0x80490db <quit>
0x8049116             add     BYTE PTR [eax],al
0x8049118             add     BYTE PTR [eax],al
0x804911a             add     BYTE PTR [eax],al
0x804911c             add     BYTE PTR [eax],al
0x804911e             add     BYTE PTR [eax],al

native process 6762 In: _start
(gdb) layout regs
(gdb) run
Starting program: /home/darfonos/work/arch-pc/lab09/lab9-5

Breakpoint 1, _start () at lab9-5.asm:11

```

Рис. 3.7: Проверка значений регистров

## Шаг 5

Перехожу к следующему брейкпоинту и снова проверяю какие значения принимают регистры. (рис. [3.8]).

- Замечаю, что умножение регистра ecx происходит на регистр eax( $4*2$ ), а к регистру ebx плюсуется 5 ( $5+5$ ) и его значение записывается в результат программы.

```

Breakpoint 2, _start () at lab9-5.asm:16
(gdb) p/s $ebx
$1 = 10
(gdb) p/s $edi
$2 = 10
(gdb) p/s $eax
$3 = 8
(gdb) 

```

Рис. 3.8: Выявление главных ошибок

## Шаг 6

Исправляю основные ошибки выявленные с помощью отладчика GDB. (рис. [3.9]).

```
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
int $0x0
```

Рис. 3.9: Исправление ошибок в программе

## Шаг 7

Создаю исполняемый файл и проверяю работу программы. (рис. [3.10]).

```
[bmavasi@fedora lab09]$ nasm -f elf -g -l lab9-5.lst lab9-5.asm
[bmavasi@fedora lab09]$ ld -m elf_i386 -o lab9-5 lab9-5.o
[bmavasi@fedora lab09]$ ./lab9-5
Результат: 25
[bmavasi@fedora lab09]$
```

Рис. 3.10: Запуск исполняемого файла

**Программа отработала без ошибок!!**

## 4 Вывод

В результате выполнения лабораторной работы, я научилась организовывать код в подпрограммы и познакомилась с базовыми функциями отладчика GDB.