

Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютера

Арфонос Дмитрий

Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Самостоятельная работа	20
3.1	Здание №1	20
3.2	Задание №2	22
4	Вывод	26

Список иллюстраций

2.1	Создание директории	5
2.2	Редактирование файла	6
2.3	Запуск исполняемого файла	7
2.4	Редоктирование	7
2.5	Запуск исполняемого файла	7
2.6	Редактирование программы	8
2.7	Создание исполняемого файла	9
2.8	Создание файла	9
2.9	Работа с отладкой gdb	10
2.10	Дисассамблирование кода	10
2.11	Синтаксис Intel	11
2.12	Режим псевдографики	12
2.13	Просмотр точек останова	13
2.14	Вывод значений регистров	14
2.15	Вывод значений переменных	15
2.16	Изменение значений переменных	15
2.17	Изменение значений переменных	16
2.18	Запуск исполняемого файла	16
2.19	Вывод значений переменных	17
2.20	Вывод значений переменных	17
2.21	копирование файла	18
2.22	Запуск исполняемого файла	18
2.23	Изменение значений переменных	18
2.24	Просмотр содержимого в esp	19
2.25	Вывод значений переменных	19
3.1	Копирование файла	20
3.2	Изменение программы	21
3.3	Запуск исполняемого файла	21
3.4	Программа вычисления выражения $(3 + 2) * 4 + 5$	22
3.5	Запуск исполняемого файла	23
3.6	Работа с отладчиком	23
3.7	Проверка значений регистров	24
3.8	Выявление главных ошибок	24
3.9	Исправление ошибок в программе	25
3.10	Запуск исполняемого файла	25

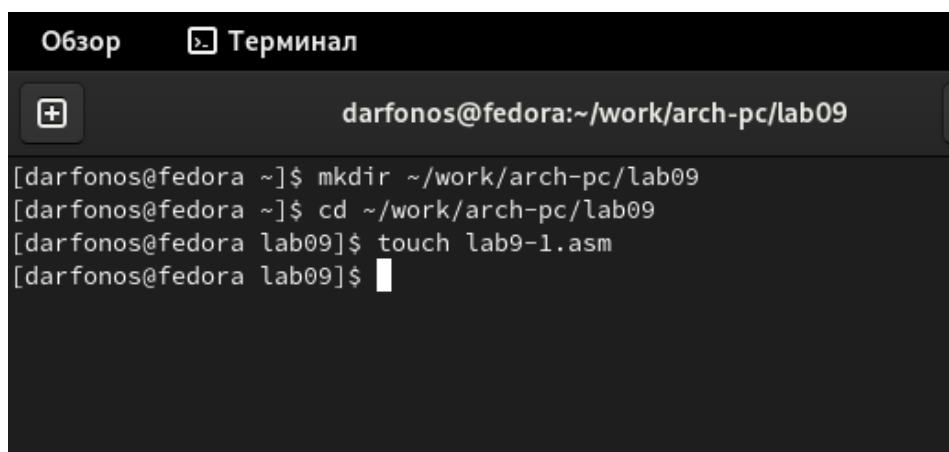
1 Цель работы

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])



```
Обзор  Терминал
darfonos@fedora:~/work/arch-pc/lab09

[darfonos@fedora ~]$ mkdir ~/work/arch-pc/lab09
[darfonos@fedora ~]$ cd ~/work/arch-pc/lab09
[darfonos@fedora lab09]$ touch lab9-1.asm
[darfonos@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
    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 ;ПОДПРОГРАММА calcul

mov eax,result
call sprint
mov eax,[res]
call iprintLF

call quit

;-----

_calcul:
    mov ebx,2
    mul ebx
    add eax,7
    mov [res],eax

    ret
```

Рис. 2.2: Редактирование файла

Шаг 3

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

```
[darfonos@fedora lab09]$ nasm -f elf lab9-1.asm
[darfonos@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[darfonos@fedora lab09]$ ./lab9-1
Введите x: 5
2x+7= 17
[darfonos@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]).

```
;-----
_calcul:
    call _subcalcul
    mov eax,[res]
    mov ebx,2
    mul ebx
    add eax,7
    mov [res],eax

    ret
_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]).

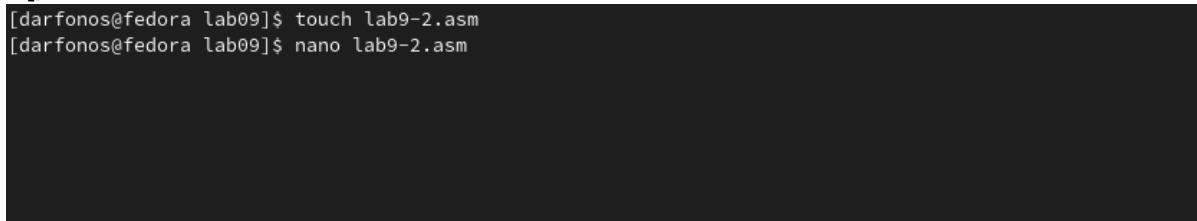
```
[darfonos@fedora lab09]$ nasm -f elf -l lab9-1.lst lab9-1.asm
[darfonos@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[darfonos@fedora lab09]$ ./lab9-1
Введите x: 5
f(x)=2x+7
g(x)= 3x-1
f(g(x)) = 35
[darfonos@fedora lab09]$
```

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. [2.6]).



```
[darfonos@fedora lab09]$ touch lab9-2.asm
[darfonos@fedora lab09]$ nano lab9-2.asm
```

{ #fig:040 width=80% }

```
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. 2.6: Редактирование программы

Шаг 7

Создаю исполняемый файл, файл листинга для работы с отладчиком GDB (рис. [2.7]).

```
[darfonos@fedora lab09]$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
[darfonos@fedora lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[darfonos@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. 2.7: Создание исполняемого файла

Шаг 8

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. [2.8]).

```
(gdb) run
Starting program: /home/darfonos/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 5563) exited normally]
(gdb) █
```

Рис. 2.8: Создание файла

Шаг 9

Для более подробного анализа программы, вставляю брэйкпоинт на метку _start (рис.[2.9]).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 11.
(gdb) run
Starting program: /home/darfonos/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:11
11      mov eax,4
(gdb)
```

Рис. 2.9: Работа с отладкой gdb

Шаг 10

Посмотрим дизассемблированный код, начиная с этой метки. (рис. [2.10]).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 11.
(gdb) run
Starting program: /home/darfonos/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:11
11      mov eax,4
(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. 2.10: Дисассамблирование кода

Шаг 11

Так же посмотрим как выглядит дизассемблированный код с синтаксисом Intel (рис. [2.11]).

```
(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. 2.11: 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 12

Включим режим псевдографики, с помощью которого отображается код программы и содержимое регистров (рис. [2.12]).

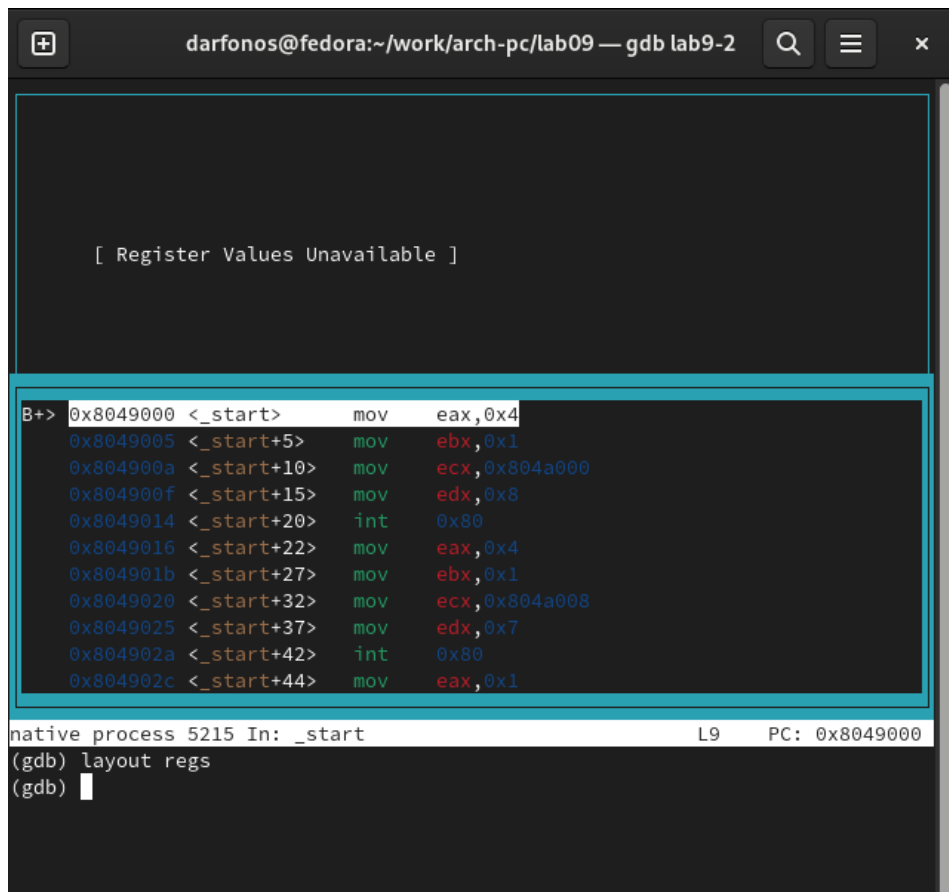


Рис. 2.12: Режим псевдографики

Шаг 13

Посмотрим информацию о наших точках останова и сразу добавим еще одну точку (рис. [2.13]).

```
[ Register Values Unavailable ]

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 5215 In: _start L9 PC: 0x8049000
(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint      keep y 0x08049000 lab9-2.asm:9
      breakpoint already hit 1 time
(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
      breakpoint already hit 1 time
2      breakpoint      keep y 0x08049031 lab9-2.asm:20
(gdb) 
```

Рис. 2.13: Просмотр точек останова

Шаг 14

Так же можно выводить значения регистров. Делается это командой `i r`. Псевдографика предствалена на (рис. [2.14]).

```
[ Register Values Unavailable ]

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 5215 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. 2.14: Вывод значений регистров

Шаг 15

В отладчике можно вывести текущее значение переменных. Сделать это можно по имени или по адресу: выводим значения переменных msg1 и msg2 (рис. [2.15]).

```

[ Register Values Unavailable ]

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 5215 In: _start L9 PC: 0x8049000
cs      0x23      35
--Type <RET> for more, q to quit, c to continue without paging--
0x2b      43
ds      0x2b      43
es      0x2b      43
fs      0x0       0
gs      0x0       0
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)

```

Рис. 2.15: Вывод значений переменных

Шаг 16

Так же отладчик позволяет менять значения переменных прямо во время выполнения программы (рис. [2.16]).

```

(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb)

```

Рис. 2.16: Изменение значений переменных

- Заменяю первый символ 'H' на 'h'

Шаг 17

Заменяя первый символ переменной `msg2` на символ `j`. (рис. [2.17]).

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

Рис. 2.17: Изменение значений переменных

Шаг 18

Выводить можно так же содержимое регистров. Выведем значение `ebx` в разных форматах: строчном, 16-ричном . (рис. [2.18]).

```
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 5215 In: _start L9 PC: 0x80490
(gdb) p/t $ebx
$2 = 110010
(gdb) p/s $ebx
$3 = 50
(gdb) p/s $ebx
$4 = 50
```

Рис. 2.18: Запуск исполняемого файла

Шаг 19

Как и переменным, регистрам можно задавать значения (рис. [2.19]).

```
(gdb) set $ebx=2
(gdb) p/s
$5 = 50
(gdb) p/s $ebx
$6 = 2
(gdb)
```

Рис. 2.19: Вывод значений переменных

Шаг 20

Так же отладчик позволяет менять значения переменных прямо во время выполнения программы (рис. [2.20]).

```
(gdb) set $ebx=2
(gdb) p/s
$5 = 50
(gdb) p/s $ebx
$6 = 2
(gdb)
```

Рис. 2.20: 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 21

Скопируем файл из лабораторной 9, переименуем её и создадим исполняемый файл.(рис. [2.21]).

```
[darfonos@fedora lab09]$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab9-3.asm
[darfonos@fedora lab09]$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
[darfonos@fedora lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
[darfonos@fedora lab09]$
```

Рис. 2.21: копирование файла

Шаг 22

Откроем отладчик и зададим аргументы. (рис. [2.22]).

```
(gdb) set args lab9-3 аргумент1 аргумент 2 'аргумент 3
(gdb)
```

Рис. 2.22: Запуск исполняемого файла

Шаг 23

Создадим точку останова на метке `_start` и запустим программу(рис. [2.23]).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/darfonos/work/arch-pc/lab09/lab9-3 lab09-3 аргумент1 а
ргумент 2 'аргумент 3'

Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 2.23: Изменение значений переменных

Шаг 24

Посмотрим на содержимое стека, что расположено по адресу, находящемуся в регистре `esp`(рис. [2.24]).

```
Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd190:    0x00000006
(gdb)
```

Рис. 2.24: Просмотр содержимого в esp

Шаг 25

Далее посмотрим на все остальные аргументы в стеке. Их адреса располагаются в 4 байтах друг от друга (именно столько занимает элемент стека) (рис. [2.25]).

```
(gdb) x/s *(void**)(esp+4)
0xffffd344:    "/home/darfonos/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp+8)
0xffffd36d:    "lab09-3"
(gdb) x/s *(void**)(esp+12)
0xffffd375:    "аргумент1"
(gdb) x/s *(void**)(esp+16)
0xffffd387:    "аргумент"
(gdb) x/s *(void**)(esp+20)
0xffffd398:    "2"
(gdb) x/s *(void**)(esp+24)
0xffffd39a:    "аргумент 3"
(gdb) x/s *(void**)(esp+28)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.25: Вывод значений переменных

3 Самостоятельная работа

3.1 Задание №1

Шаг 1

Копирую программу из лабораторной 8 (рис. [3.1]).

```
Quit anyway? (y or n) y
[darfonos@fedora lab09]$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab9-4.asm
[darfonos@fedora lab09]$
```

Рис. 3.1: Копирование файла

Шаг 2

Изменяю текст программы с использованием подпрограмм (рис. [3.2]).

```

next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
call _calcul ; вызов подпрограммы
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax,msg1 ;
call printf ;
mov eax, msg ; вывод сообщения "Результат: "
call printf
mov ecx, esi ; записываем сумму в регистр `ecx`
call printf ; печать результата
call _quit ; завершение программы
;-----
_calcul:
mov ebx,3 ;ebx=3
mul ebx; ecx=ecx*ebx
sub ecx,1 ; ecx-1
add esi,ecx ; добавляем к промежуточной сумме
;----возврат в основную программу
ret

```

Рис. 3.2: Изменение программы

Шаг 3

Создаю исполняемый файл и проверяю работу изменённой программы (рис. [3.3]).

```

[darfonos@fedora lab09]$ nasm -f elf lab9-4.asm
[darfonos@fedora lab09]$ ld -m elf_i386 -o lab9-4 lab9-4.o
[darfonos@fedora lab09]$ ./lab9-4 4 3 2 1
Функция: f(x)=3x-1
Результат: 26
[darfonos@fedora lab09]$

```

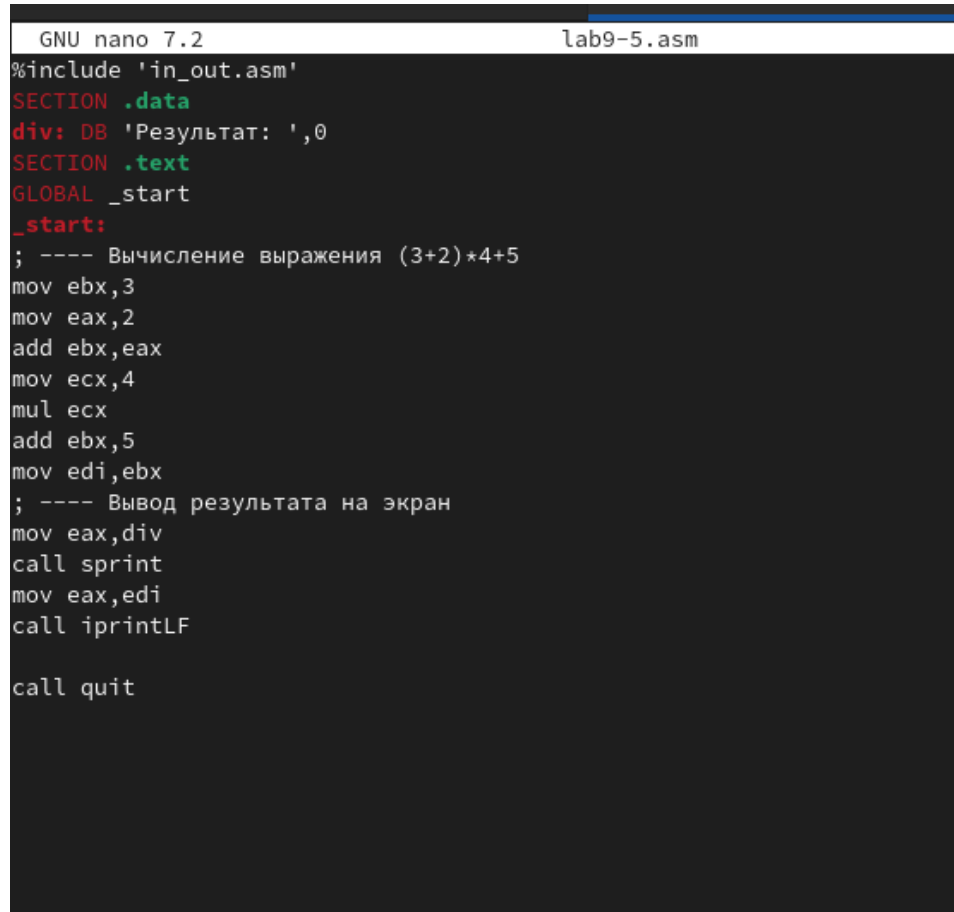
Рис. 3.3: Запуск исполняемого файла

Программа отработала верно

3.2 Задание №2

Шаг 1

Создаю новый файл и вставляю в него программу из листинга (рис. [3.4]).



```
GNU nano 7.2                                lab9-5.asm
%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. 3.4: Программа вычисления выражения $(3 + 2) * 4 + 5$

Шаг 2

Проверяю работу программы и вижу, что в его тексте есть ошибки (рис. [3.5]).

```
[darfonos@fedora lab09]$ nasm -f elf -g -l lab9-5.lst lab9-5.asm
[darfonos@fedora lab09]$ ld -m elf_i386 -o lab9-5 lab9-5.o
[darfonos@fedora lab09]$ ./lab9-5
Результат: 10
[darfonos@fedora lab09]$
```

Рис. 3.5: Запуск исполняемого файла

Шаг 3

Далее открываю программу в отладчике. Для того, чтобы найти ошибку дисасемблирую программу и добавляю брейкпоинты в основной части программы (рис. [3.6]).

```
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
   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 *0x080490fe
Breakpoint 2 at 0x80490fe: file lab9-5.asm, line 14.
(gdb) run
```

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]).

```
Breakpoint 1, _start () at lab9-5.asm:11
11      mov ecx,4
(gdb) p/s $ebx
$1 = 5
(gdb) p/s $eax
$2 = 2
(gdb)
```

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.

```
(gdb) c
Continuing.

Breakpoint 2, _start () at lab9-5.asm:14
14      mov edi,ebx
(gdb) p/s $eax
$3 = 8
(gdb) p/s $ebx
$4 = 10
(gdb) p/s $ecx
$5 = 4
(gdb)
```

Рис. 3.8: Выявление главных ошибок

Шаг 6

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

```
mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
```

Рис. 3.9: Исправление ошибок в программе

Шаг 7

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

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

Рис. 3.10: Запуск исполняемого файла

Программа отработала без ошибок!!

4 Вывод

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