РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 9

дисциплина: Архитектура компьютера

Студент: Мурылев Иван Валерьевич

Группа: НПИбд-03-25

МОСКВА

2025 г.

Содержание

Цель работы

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

Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе.

Отладка делится на 4 этапа :

обнаружение ошибки;

• поиск её местонахождения;

• определение причины ошибки;

• исправление ошибки.

Методы отладки.

Наиболее часто применяют следующие методы отладки:

• создание точек контроля значений на входе и выходе участка программы (например,вывод промежуточных значений на экран — так называемые диагностические сообщения);

• использование специальных программ-отладчиков.

Есть два самых популярных способа отладки.

Пошаговое выполнение — это выполнение программы с остановкой после каждой строчки, чтобы программист мог проверить значения переменных и выполнить другие действия.

Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова:

• Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом);

• Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его).

Отладчик GDB

Отладчик **GDB** (как и любой другой отладчик) позволяет увидеть, что происходит «внутри»
программы в момент её выполнения или что делает программа в момент сбоя.

GDB может выполнять следующие действия:

• начать выполнение программы, задав всё, что может повлиять на её поведение;

• остановить программу при указанных условиях;

• исследовать, что случилось, когда программа остановилась;

• изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

Выполнение работы

Лабараторая часть

Так как в структуре репозитория уже создана папка отчета, первый этап пропускается.

После создания файл lab09-1.asm проверяем его наличие.

```
[ivmurihlev@personal ~]$ cd Documents/work/study/2025-2026/arch_Evm/arch-evm/labs/lab09/report/
[ivmurihlev@personal report]$ touch lab09-1.asm
[ivmurihlev@personal report]$ mc

[ivmurihlev@personal report]$ ls lab09-1.asm
lab09-1.asm
```

Заносим Листинг 9.1 в 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:
;---------------------------------------------
; 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"
_calcul:
mov ebx,2
mul ebx
add eax,7
102 Демидова А. В.
mov [res],eax
ret ; выход из подпрограммы
```

Транслируем и создаем исполняемый файл и запускаем его.

```
[ivmurihlev@personal report]$ nasm -f elf lab09-1.asm
[ivmurihlev@personal report]$ ld  -m  elf_i386  lab09-1.o -o lab09-1
[ivmurihlev@personal report]$ ./lab09-1
Введите x: 45
2x+7=97
```

Затем изменяем его согласно заданию (добавив подпрограмму _subcalcul в подпрограмму _calcul, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. Т.е. x передается в подпрограмму _calcul из нее в подпрограмму _subcalcul, где вычисляется выражение $g(x)$, результат возвращается в _calcul и вычисляется выражение$f(g(x))$.) добавляя модуль как показано ниже:

```
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _subcalcul
mov eax,[gx]
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы


_subcalcul:
mul eax,3
add eax,-1
mov [gx],eax
ret
```

После этого прохоим процедуру создания исполняемого файла и запускаем его вводя 3 и 1 соответственно.

```
[ivmurihlev@personal report]$ nasm -f elf lab09-1.asm
[ivmurihlev@personal report]$ ld  -m  elf_i386  lab09-1.o -o lab09-1
[ivmurihlev@personal report]$ ./lab09-1
Введите x: 3
2x+7=23
[ivmurihlev@personal report]$ ./lab09-1 1
Введите x: 1
2x+7=11
```

Как видно программа работает верно.

Создадим файл lab09-2.asm и внесем в него.

```
[ivmurihlev@personal report]$ ls
arch-pc--lab09--report.qmd  bib    in_out.asm  lab09-1.asm  lab09-2.asm
_assets                     image  lab09-1     lab09-1.o    Makefile
[ivmurihlev@personal report]$ cat lab09-2.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
```

Затем транслируем и создаем исполяемый файл как впримере.

```
[ivmurihlev@personal report]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[ivmurihlev@personal report]$ ld -m elf_i386 -o lab09-2 lab09-2.o
[ivmurihlev@personal report]$ ls
arch-pc--lab09--report.qmd  _assets  bib  image  in_out.asm  lab09-1  lab09-1.asm  lab09-1.o  lab09-2  lab09-2.asm  lab09-2.lst  lab09-2.o
```

После этого запускаем gdb и создаем точку останова _start

```
Reading symbols from lab09-2...
(gdb) run
Starting program: /home/ivmurihlev/Documents/work/study/2025-2026/arch_Evm/arch-evm/labs/lab09/report/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.archlinux.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.
Hello, world!
[Inferior 1 (process 214477) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/ivmurihlev/Documents/work/study/2025-2026/arch_Evm/arch-evm/labs/lab09/report/lab09-2

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

Затем вводим команду disassemble _start

```
Breakpoint 1, _start () at lab09-2.asm:9
9       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.
```

Теперь меняем синтаксис на Intel’овский через предложенную команду и посмотрим вывод:

```
(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.
```

Подметим их отличия (ATT и Intel)

1. У ATT перед регистром стоит знак % , а у Intel нет.
2. **Перед константой в ATT стоит $**

После обращения к регистрам получаем:

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

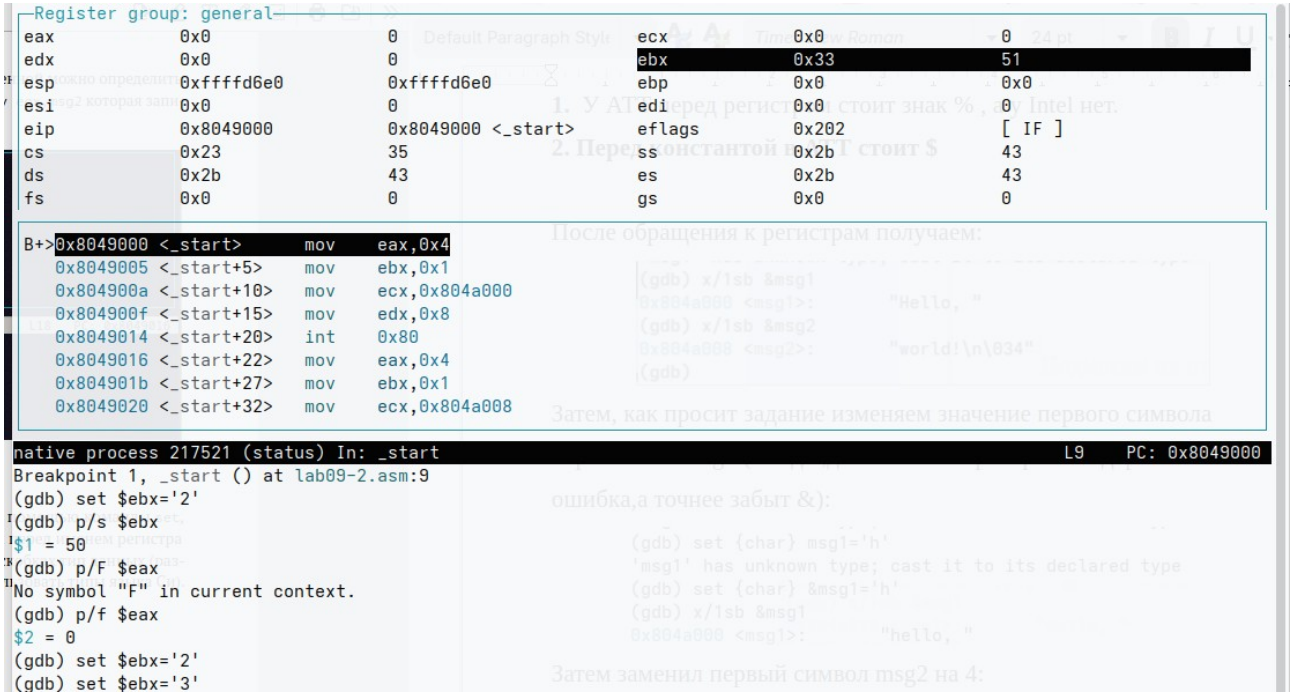
Затем, как просит задание изменяем значение первого символа переменной msg1 (в коде ,данном в лабароторной содержится ошибка,а точнее забыт &):

```
(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, "
```

Затем заменил первый символ msg2 на 4:

```
0x804a008 <msg2>:       "4orld!\n\034"
(gdb) x/1sb &msg2
```

Далее меняем регитры



Как видно значение регистра меняется в соответствии с заданым значением, а именно номером символа.

Создаем файл lab09-3.asm копируем в не файл lab08-2.asm.

```
[ivmurihlev@personal report]$ touch lab09-3.asm
[ivmurihlev@personal report]$ cp lab08-2.asm lab09-3.asm
[ivmurihlev@personal report]$ cat lab09-3.asm
%include 'in_out.asm'
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
next:
cmp ecx, 0 ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем аргумент из стека
call sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

Затем создадим исполняемый файл:

```
[ivmurihlev@personal report]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[ivmurihlev@personal report]$ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Запускаем отладку указав аргументы:

```
[ivmurihlev@personal report]$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) 16.3
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-pc-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...
```

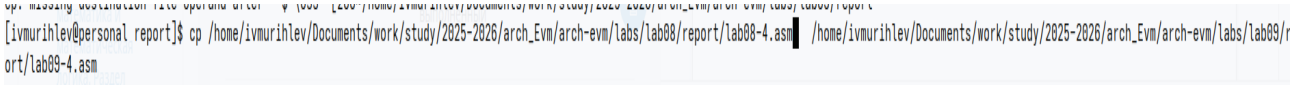
Выполняем просмотр позиций стека:

Так как размер указателя равен 4 байтам (в x86 архитектуре),то и смещение будет 4. И поэтому адресс будет равен [esm(начало) +N(номер в стеке)*4].

Самостоятельная часть

Задание 1

Копирую файл программы из самостоятельной работы №8 (lab08-4.asm) в lab09-4.asm:



После всенения требуемого изменения:

```
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
call _function
mov eax,msg2

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

_function:
mov ebx,3
add eax,2
mul ebx
mov [res],eax
ret
```

Затем создаю и запускаю исполняемый файл:

```
[ivmurihlev@personal report]$ nasm -f elf lab09-4.asm
[ivmurihlev@personal report]$ ld  -m  elf_i386  lab09-4.o -o lab09-4
[ivmurihlev@personal report]$ ./lab09-4
Функция 3(x+2)
Segmentation fault         (core dumped) ./lab09-4
[ivmurihlev@personal report]$ ./lab09-4 3
Функция 3(x+2)
Результат: 15
[ivmurihlev@personal report]$ ./lab09-4 4
Функция 3(x+2)
Результат: 18
[ivmurihlev@personal report]$ ./lab09-4 45
Функция 3(x+2)
Результат: 141
```

Как видно программа работает корректно.

Задание 2

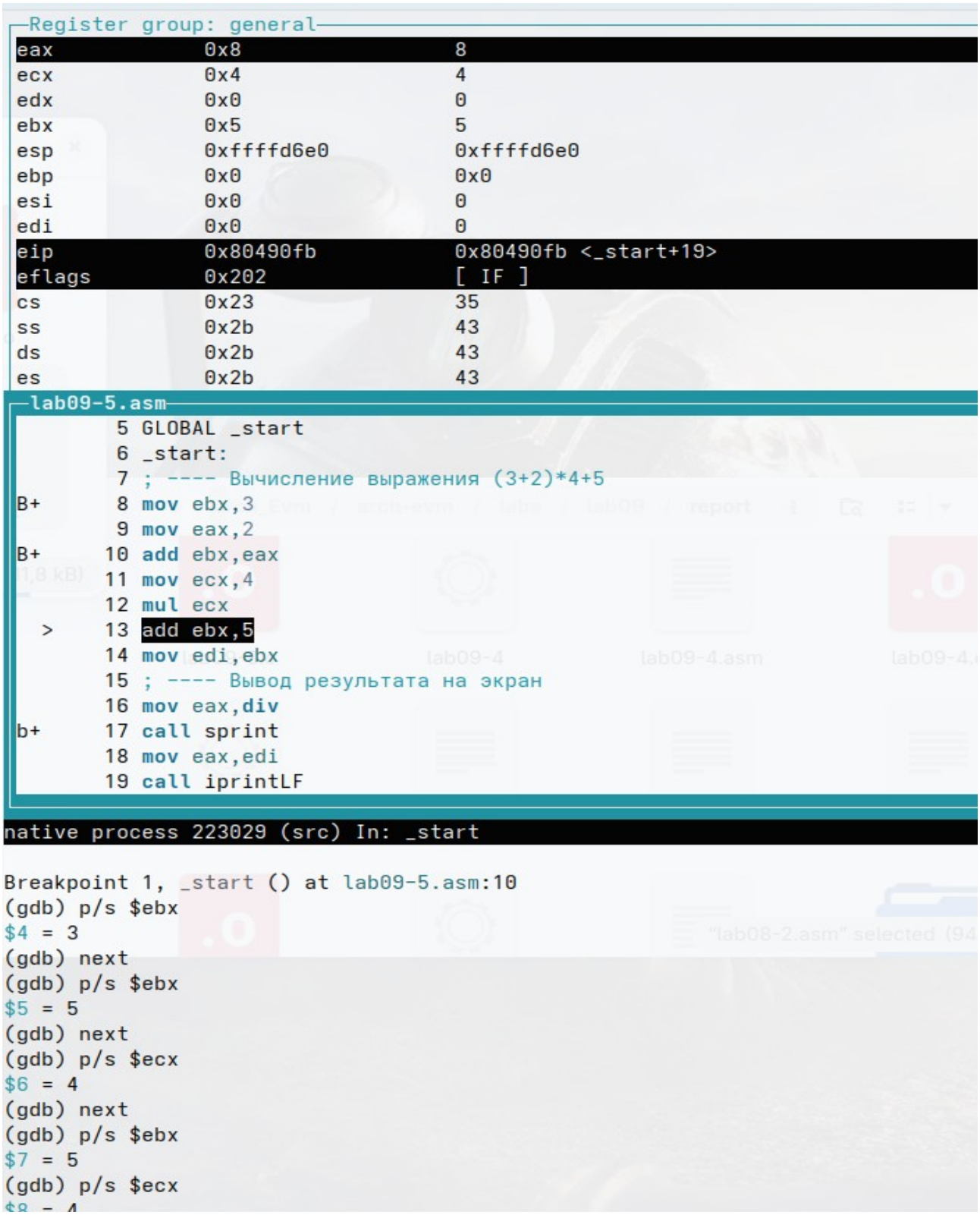
После создания и заполнение транслируем его для отладки.

```
[ivmurihlev@personal report]$ cat lab09-5.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
[ivmurihlev@personal report]$ nasm -f elf -g -l lab09-5.lst lab09-5.asm
```

Создадим объектный файл и проверим его работу:

```
[ivmurihlev@personal report]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[ivmurihlev@personal report]$ ld -m elf_i386 -o lab09-5 lab09-5.o
[ivmurihlev@personal report]$ ./lab09-5
Результат: 10
```

В отладке ввидно:



Что ecx (4) при умнажении идет к первому регистру eax.Поменяв код на следующий:

```
%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 eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Потом транслирую и запускаю и получаю корректно работующий код:

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

Вывод

Изучена удобность подпрограмм и особенности отладки через gdb, а также его необычная способность задать значение прямо в момент отладки , что не наблюдается в отладке языков более высокого уровня.Был изучен интерфейс gdb.