

# **Отчёт по лабораторной работе №9**

**Архитектура компьютера**

Андреева Софья Владимировна

# Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Задание для самостоятельной работы.	15
4	Выводы	20

# Список иллюстраций

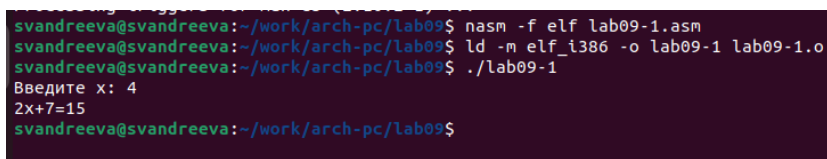
2.1	Результат работы файла lab09-1.asm . . . . .	5
2.2	Измененный файл . . . . .	6
2.3	Запуск измененного файла . . . . .	6
2.4	Результат работы файла lab09-2.asm . . . . .	7
2.5	Брейкпоинт . . . . .	7
2.6	Дисассимилированный код программы . . . . .	8
2.7	Отображение команд с Intel'овским синтаксисом . . . . .	8
2.8	Режим псевдографики . . . . .	9
2.9	info breakpoints . . . . .	9
2.10	Установка точки останова . . . . .	9
2.11	изменение значений регистров . . . . .	10
2.12	Значения переменных msg1 и msg2 . . . . .	10
2.13	Команда set . . . . .	11
2.14	Значения регистра edx . . . . .	11
2.15	Значения регистра ebx . . . . .	12
2.16	Работа файла lab9-3.asm . . . . .	13
2.17	Адрес вершины стека . . . . .	13
2.18	Все позиции стека . . . . .	14
3.1	Программа вычисления значения функции . . . . .	17
3.2	Запуск файла . . . . .	17
3.3	Поиск ошибок . . . . .	18
3.4	Запуск файла . . . . .	19

# 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

Создадим каталог для программ лабораторной работы № 9, перейдем в него и создадим файл lab09-1.asm. Введем в файл lab09-1.asm текст программы из листинга 9.1. Создадим исполняемый файл и запустим его (рис. 2.1).



```
svandreeva@svandreeva:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
svandreeva@svandreeva:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
svandreeva@svandreeva:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 4
2x+7=15
svandreeva@svandreeva:~/work/arch-pc/lab09$
```

Рис. 2.1: Результат работы файла lab09-1.asm

Изменим программу, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры,  $f(x)=2x+7$ ,  $g(x)=3x-1$ . Переменная  $x$  передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение  $g(x)$ , результат возвращается в `_calcul` и вычисляется выражение  $f(g(x))$ . Результат возвращается в основную программу для вывода результата на экран. (рис. 2.2).

```

call _printf
mov eax,[rez]
call _printf
call quit
;-----
; Подпрограмма вычисления
; выражения "f(g(x))"
_calcul:
call _subcalcul ; сначала вычисляем g(x)
mov ebx,2
mul ebx
add eax,7
mov [rez],eax
ret ; выход из подпрограммы

;-----
; Подпрограмма вычисления
; выражения "g(x)=3x-1"
_subcalcul:
mov ebx, 3
mul ebx
dec eax
ret

```

Рис. 2.2: Измененный файл

Создадим исполняемый файл и запустим его. Всё получилось (рис. 2.3).

```

svandreeva@svandreeva:~/work/arch-pc/lab09$ mc
svandreeva@svandreeva:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
svandreeva@svandreeva:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
svandreeva@svandreeva:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
f(x)=2x+7, g(x)= 3x-1, f(g(x)) = 2(3x-1)+7 = 23
svandreeva@svandreeva:~/work/arch-pc/lab09$

```

Рис. 2.3: Запуск измененного файла

Создадим файл lab09-2.asm и введем в него текст программы из листинга 9.2. Получим исполняемый файл, добавив для работы с GDB отладочную информацию, трансляцию программ проводим с ключом -g. Загрузим исполняемый файл в отладчик gdb и проверим работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 2.4).

```

svandreeva@svandreeva:~/work/arch-pc/lab09$ touch lab09-2.asm
svandreeva@svandreeva:~/work/arch-pc/lab09$ mc
svandreeva@svandreeva:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
svandreeva@svandreeva:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
svandreeva@svandreeva:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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".
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) run
Starting program: /home/svandreeva/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 3524) exited normally]
(gdb)

```

Рис. 2.4: Результат работы файла lab09-2.asm

Для более подробного анализа программы установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её.(рис. 2.5).

```

[Inferior 1 (process 3524) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/svandreeva/work/arch-pc/lab09/lab09-2

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

```

Рис. 2.5: Брейкпоинт

Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`(рис. 2.6).

```

(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. 2.6: Дисассимилированный код программы

Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`(рис. 2.7).

```

(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) █

```

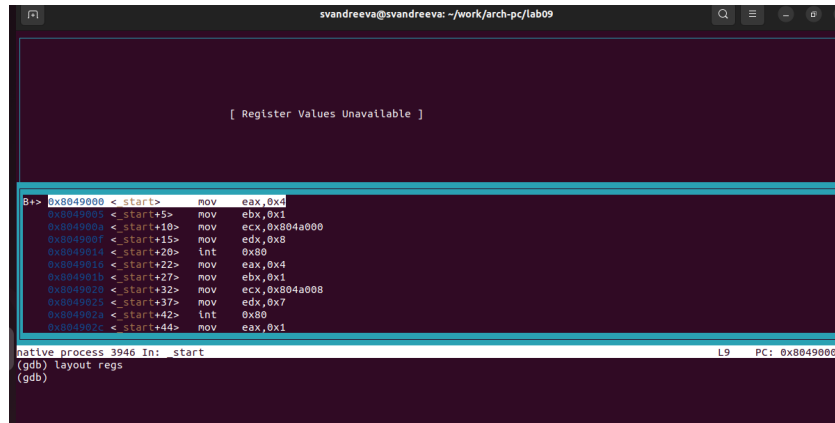
Рис. 2.7: Отображение команд с Intel'овским синтаксисом

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel: противоположное расположение операнда-источника и операнда-приемника (в Intel - приемник, источник, а в АТТ - источник, приемник); в АТТ регистры пишутся после '%', а непосредственные операнды после '\$', в синтаксисе Intel



операнды никак не помечаются.

Включим режим псевдографики для более удобного анализа программы(рис. 2.8).



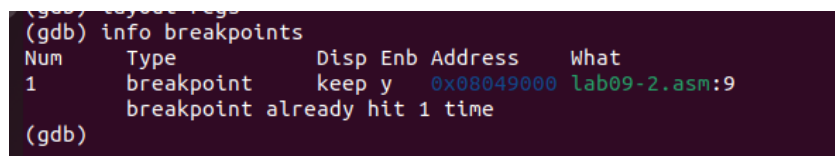
```
[ Register Values Unavailable ]

0x08049000 < start>    mov     eax, 0x4
0x08049001 < start+5>    mov     ebx, 0x1
0x0804900a < start+10>   mov     ecx, 0x04a000
0x0804900f < start+15>   mov     edx, 0x8
0x08049014 < start+20>   int     0x0
0x08049016 < start+22>   mov     eax, 0x4
0x0804901b < start+27>   mov     ebx, 0x1
0x08049020 < start+32>   mov     ecx, 0x04a008
0x08049025 < start+37>   mov     edx, 0x7
0x0804902a < start+42>   int     0x0
0x0804902c < start+44>   mov     eax, 0x1

native process 3946 In: start
(gdb) layout regs
(gdb)
```

Рис. 2.8: Режим псевдографики

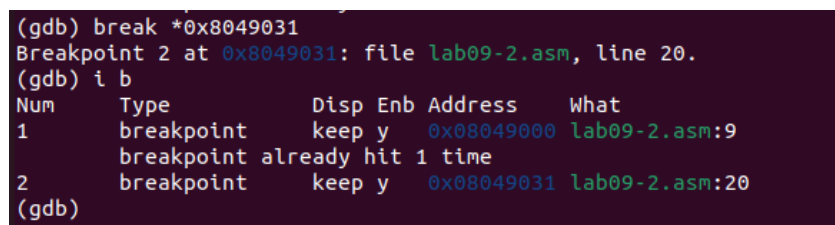
На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверим это с помощью команды `info breakpoints`(рис. 2.9) .



```
(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. 2.9: info breakpoints

Установим еще одну точку останова по адресу предпоследней инструкции.Посмотрим информацию о всех установленных точках останова(рис. 2.10).



```
(gdb) break *0x08049031
Breakpoint 2 at 0x08049031: 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)
```

Рис. 2.10: Установка точки останова

Выполним 5 инструкций с помощью команды stepi и проследим за изменением значений регистров. В результате изменяются значения регистров eax, ebx, ecx, edx (рис. 2.11).

```
svandreeva@svandreeva: ~/work/arch-pc/lab09
Register group: general
eax      0x8      8      ecx      0x804a000    134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffd150 0xffffd150  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
ebp      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+ 0x8049008 < _start> mov $0x4,%eax
0x8049005 < _start+5> mov $0x1,%ebx
0x804900a < _start+10> mov $0x804a000,%ecx
0x804900f < _start+15> mov $0x8,%edx
0x8049014 < _start+20> int $0x80
> 0x8049016 < _start+22> mov $0x4,%eax
0x804901b < _start+27> mov $0x1,%ebx
0x8049020 < _start+32> mov $0x804a000,%ecx
0x8049025 < _start+37> mov $0x7,%edx
0x804902a < _start+42> int $0x80
0x804902c < _start+44> mov $0x1,%eax
b+ 0x8049031 < _start+49> mov $0x0,%ebx

native process 4532 In: start
breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) si
The program is not being run.
(gdb) run
starting program: /home/svandreeva/work/arch-pc/lab09/lab09-2
breakpoint 1, _start () at lab09-2.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рис. 2.11: изменение значений регистров

Посмотрели значения переменных msg1 и msg2, причем адрес памяти msg1 задали по имени переменной, а адрес памяти msg2 вписали вручную, взяв его из дисассемблированного кода программы (рис. 2.12).

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

Рис. 2.12: Значения переменных msg1 и msg2

С помощью команды set заменили в msg1 первый символ, в переменной msg2 - второй (рис. 2.13).

```

(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}0x804a009 = '0'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "w0rld!\n\034"
(gdb)

```

Рис. 2.13: Команда set

Выведем значения регистра `edx` в десятичном, шестнадцатеричном и двоичном формате, соответственно (рис. 2.14).

```

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

```

Рис. 2.14: Значения регистра `edx`

Используя команду `set` изменили значение регистра `ebx` сначала на символ '2', а затем на число 2, и сравнили вывод значения регистра в десятичном формате. В результате присвоения регистра значение символа '2', выводится число 50, что соответствует символу в '2' в таблице ASCII (рис. 2.15).

```
73  = 1000  
(gdb) set $ebx = '2'  
(gdb) p/s $ebx  
$4 = 50  
(gdb) set $ebx = 2  
(gdb) p/s $ebx  
$5 = 2  
(gdb) █
```

Рис. 2.15: Значения регистра ebx

Затем завершили выполнение программы командой `continue` (сокращенно `c`), и вышли из GDB по команде `quit`.

Для примера обработки аргументов командной строки в GDB скопировали файл `lab8-2.asm` из Лабораторной работы №8 (программа выводит на экран аргументы командной строки) в файл `lab9-3.asm`, создали исполняемый файл и запустили в GDB с ключом `—args`. Установили точку останова перед первой инструкцией в программе и запустили ее (рис. 2.16).

```

svandreeva@svandreeva:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
svandreeva@svandreeva:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
svandreeva@svandreeva:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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".
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) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/svandreeva/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. 2.16: Работа файла lab9-3.asm

Проверила адрес вершины стека и убедилась что там хранится 5 элементов (рис. 2.17).

```

(gdb) x/x $esp
0xffffd100: 0x00000005
(gdb)

```

Рис. 2.17: Адрес вершины стека

Я посмотрела все позиции стека. В первом хранится адрес, в остальных хранятся элементы. Элементы расположены с интервалом в 4 единицы, так как стек может хранить до 4 байт: каждый элемент стека занимает 4 байта, поэтому для получения следующего элемента стека мы добавляем 4 к адресу вершины (рис. 2.18).

```
(gdb) x/s *(void**)(esp + 4)
0xffffd2de:  "/home/svandreeva/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd30a:  "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd31c:  "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd32d:  "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd32f:  "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:  <error: Cannot access memory at address 0x0>
(gdb) █
```

Рис. 2.18: Все позиции стека

### 3 Задание для самостоятельной работы.

1. Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму. Запустим файл. Все исполнилось корректно (рис. 3.1).

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg1 db "Функция:  $f(x)=15*x-9$  ",0
```

```
msg2 db "Результат: ",0
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
mov eax, msg1
```

```
call sprintLF
```

```
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)
```

```

mov esi, 0 ; Используем `esi` для хранения
; промежуточных сумм
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 ; вызываем подпрограмму для вычисления f(x)

add esi, eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg2 ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы

_calcul: ; подпрограмма для вычисления f(x)=15*x-9
mov ebx, 15
mul ebx
sub eax, 9
ret

```



```

svandreeva@svandreeva:~/work/arch-pc/lab09$ nasm -f elf samr12.asm
svandreeva@svandreeva:~/work/arch-pc/lab09$ ld -m elf_i386 -o samr12 samr12.o
svandreeva@svandreeva:~/work/arch-pc/lab09$ ./samr12
Функция: f(x)=15*x-9
Результат: 0
svandreeva@svandreeva:~/work/arch-pc/lab09$ ./samr12 3 3 2 2
Функция: f(x)=15*x-9
Результат: 114
svandreeva@svandreeva:~/work/arch-pc/lab09$

```

Рис. 3.1: Программа вычисления значения функции

2.Создадим файл с текстом из листинга 9.3, где приведена программа вычисления выражения  $(3 + 2) * 4 + 5$ . При запуске данная программа дает неверный результат(рис. 3.2).

```

svandreeva@svandreeva:~/work/arch-pc/lab09$ touch samr22.asm
svandreeva@svandreeva:~/work/arch-pc/lab09$ mc
svandreeva@svandreeva:~/work/arch-pc/lab09$ nasm -f elf samr22.asm
svandreeva@svandreeva:~/work/arch-pc/lab09$ ld -m elf_i386 -o samr22 samr22.o
svandreeva@svandreeva:~/work/arch-pc/lab09$ ./samr22
Результат: 10
svandreeva@svandreeva:~/work/arch-pc/lab09$

```

Рис. 3.2: Запуск файла

Проанализировали с помощью отладчика GDB изменение значений регистров, чтобы найти ошибку. Установили брейкпоинт `b _start`, включили режим псевдографики и начали поочередно выполнять команды и следить за значениями регистров. Первая ошибка - результат  $3+2=5$  записан в регистре `ebx` (команда `add ebx, eax`), но команда `mul` всегда перемножает значение регистра `eax` с указанным сомножителем, поэтому `mul esx` дает неверный результат:  $(3+2)*4=5$ . Затем на убеждении, что регистр `ebx` содержит корректный результат вычислений, к нему добавляется число 5, результат запоминается в регистре `edi`, а затем выводится (рис. 3.3).

```

svandreeva@svandreeva: ~/work/arch-pc/lab09
Register group: general
eax      0x8      0      ecx      0x4      4
edx      0x0      0      ebx      0x5      5
esp      0xffffd150 0xffffd150  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x80490fb 0x80490fb < start+19>  eflags   0x202    [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43
fs       0x0      0      gs       0x0      0

B+ 0x80490e8 < start>    mov     $0x3,%ebx
0x80490ed < start+5>    mov     $0x2,%eax
0x80490f2 < start+10>   add     %eax,%ebx
0x80490f4 < start+12>   mov     $0x4,%ecx
0x80490f9 < start+17>   mul     %ecx
> 0x80490fb < start+19> add     $0x5,%ebx
0x80490fe < start+22>   mov     %ebx,%edi
0x8049100 < start+24>   mov     $0x04a000,%eax
0x8049105 < start+29>   call   0x804900f <sprint>
0x804910a < start+34>   mov     %edi,%eax
0x804910c < start+36>   call   0x8049086 <|printLF>
0x8049111 < start+41>   call   0x80490db <quit>

native process 6012 In: _start
(gdb) si
(gdb) layout regs
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/svandreeva/work/arch-pc/lab09/sanr22

Breakpoint 1, _start () at sanr22.asm:8
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 3.3: Поиск ошибок

Исправим текст программы:

```
%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
```

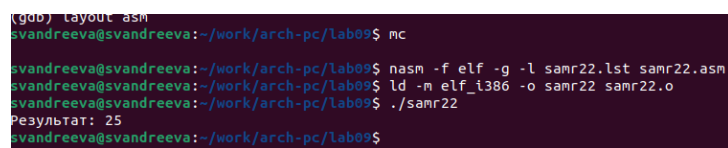
```
mov ecx,4
mul ecx
add eax,5
mov edi,eax
```

; — Вывод результата на экран

```
mov eax,div
call sprint
mov eax,edi
call iprintLF

call quit
```

Запустим файл. Все выполнилось корректно. (рис. 3.4).



```
(gdb) layout asm
svandreeva@svandreeva:~/work/arch-pc/lab09$ mc
svandreeva@svandreeva:~/work/arch-pc/lab09$ nasm -f elf -g -l samr22.lst samr22.asm
svandreeva@svandreeva:~/work/arch-pc/lab09$ ld -m elf_i386 -o samr22 samr22.o
svandreeva@svandreeva:~/work/arch-pc/lab09$ ./samr22
Результат: 25
svandreeva@svandreeva:~/work/arch-pc/lab09$
```

Рис. 3.4: Запуск файла

## 4 Выводы

Я приобрела навыки написания программ с использованием подпрограмм и познакомилась с методами отладки при помощи GDB и его основными возможностями.