

# Отчёт по лабораторной работе 9

Архитектура компьютера

Гуламова Е.М. НПИбд-03-23

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	27

## Список иллюстраций

2.1	Программа в файле lab9-1.asm . . . . .	7
2.2	Запуск программы lab9-1.asm . . . . .	7
2.3	Программа в файле lab9-1.asm . . . . .	8
2.4	Запуск программы lab9-1.asm . . . . .	9
2.5	Программа в файле lab9-2.asm . . . . .	10
2.6	Запуск программы lab9-2.asm в отладчике . . . . .	11
2.7	Дизассимилированный код . . . . .	12
2.8	Дизассимилированный код в режиме интел . . . . .	13
2.9	Точка остановки . . . . .	14
2.10	Изменение регистров . . . . .	15
2.11	Изменение регистров . . . . .	16
2.12	Изменение значения переменной . . . . .	17
2.13	Вывод значения регистра . . . . .	18
2.14	Вывод значения регистра . . . . .	19
2.15	Вывод значения регистра . . . . .	20
2.16	Программа в файле lab9-4.asm . . . . .	21
2.17	Запуск программы lab9-4.asm . . . . .	22
2.18	Код с ошибкой . . . . .	23
2.19	Отладка . . . . .	24
2.20	Код исправлен . . . . .	25
2.21	Проверка работы . . . . .	26

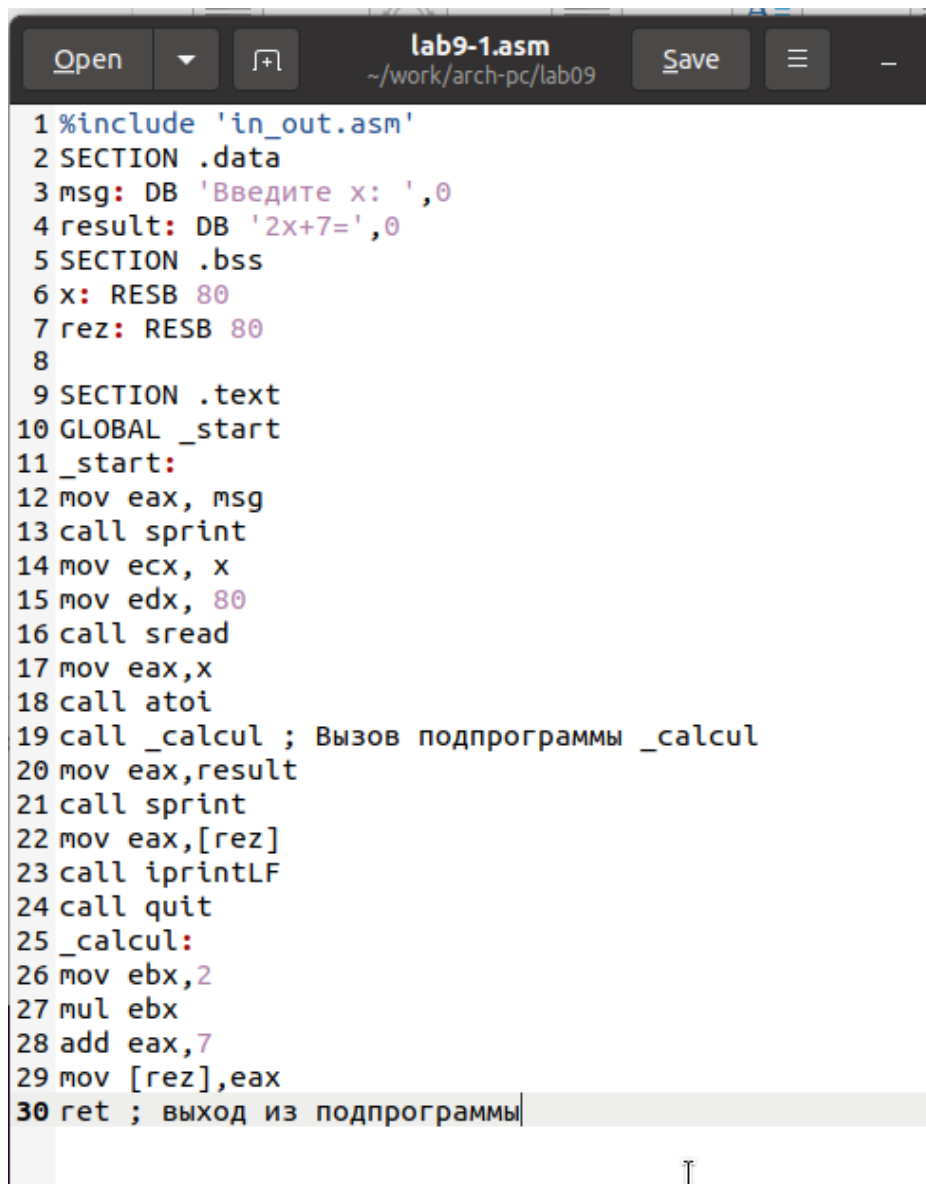
## Список таблиц

# 1 Цель работы

Целью работы является приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

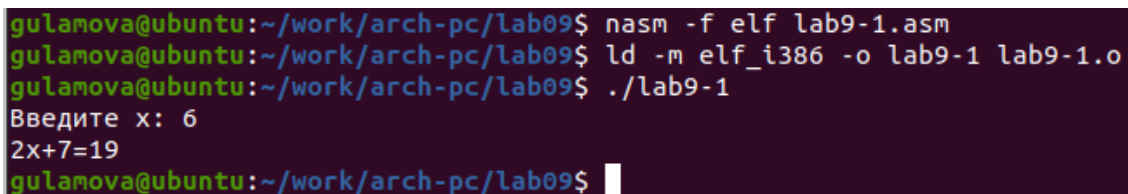
## 2 Выполнение лабораторной работы

1. Я создала папку для выполнения лабораторной работы номер девять, затем перешла в неё и сформировала файл lab9-1.asm.
2. Давайте рассмотрим пример программы, которая вычисляет арифметическую функцию  $f(x) = 2x + 7$  с использованием вспомогательной подпрограммы calcul. В этом случае значение  $x$  мы получаем через ввод с клавиатуры, а расчёт самой функции происходит внутри подпрограммы.



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 rez: RESB 80
8
9 SECTION .text
10 GLOBAL _start
11 _start:
12 mov eax, msg
13 call sprint
14 mov ecx, x
15 mov edx, 80
16 call sread
17 mov eax, x
18 call atoi
19 call _calcul ; Вызов подпрограммы _calcul
20 mov eax, result
21 call sprint
22 mov eax, [rez]
23 call iprintLF
24 call quit
25 _calcul:
26 mov ebx, 2
27 mul ebx
28 add eax, 7
29 mov [rez], eax
30 ret ; выход из подпрограммы
```

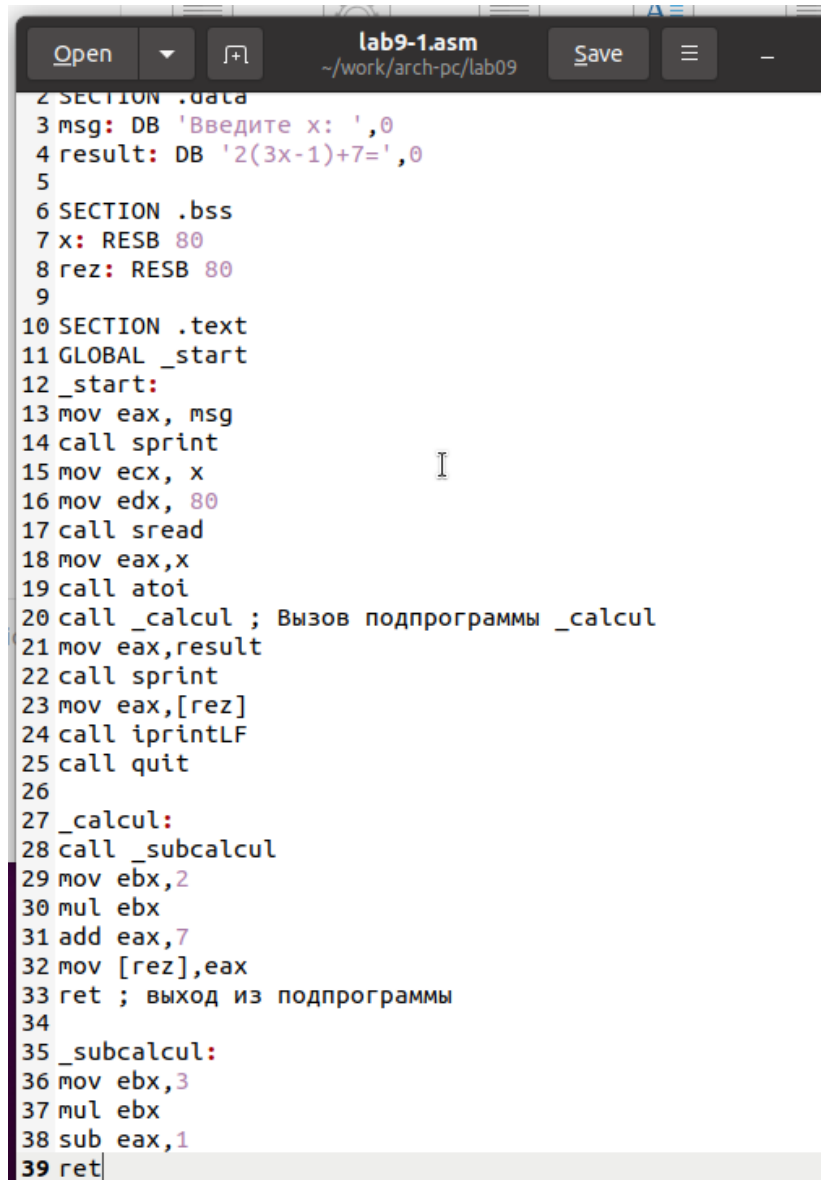
Рис. 2.1: Программа в файле lab9-1.asm



```
gulamova@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
gulamova@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
gulamova@ubuntu:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 6
2x+7=19
gulamova@ubuntu:~/work/arch-pc/lab09$
```

Рис. 2.2: Запуск программы lab9-1.asm

3. Я внесла изменения в код программы, добавив в подпрограмму `calcul` дополнительную подпрограмму `subcalcul`. Это позволило мне вычислить составное выражение  $f(g(x))$ , где  $x$  также вводится через клавиатуру, а функции заданы как  $f(x) = 2x + 7$  и  $g(x) = 3x - 1$ .



```
lab9-1.asm
~/work/arch-pc/lab09
Open Save

2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2(3x-1)+7=',0
5
6 SECTION .bss
7 x: RESB 80
8 rez: RESB 80
9
10 SECTION .text
11 GLOBAL _start
12 _start:
13 mov eax, msg
14 call sprint
15 mov ecx, x
16 mov edx, 80
17 call sread
18 mov eax, x
19 call atoi
20 call _calcul ; Вызов подпрограммы _calcul
21 mov eax, result
22 call sprint
23 mov eax, [rez]
24 call iprintLF
25 call quit
26
27 _calcul:
28 call _subcalcul
29 mov ebx, 2
30 mul ebx
31 add eax, 7
32 mov [rez], eax
33 ret ; выход из подпрограммы
34
35 _subcalcul:
36 mov ebx, 3
37 mul ebx
38 sub eax, 1
39 ret
```

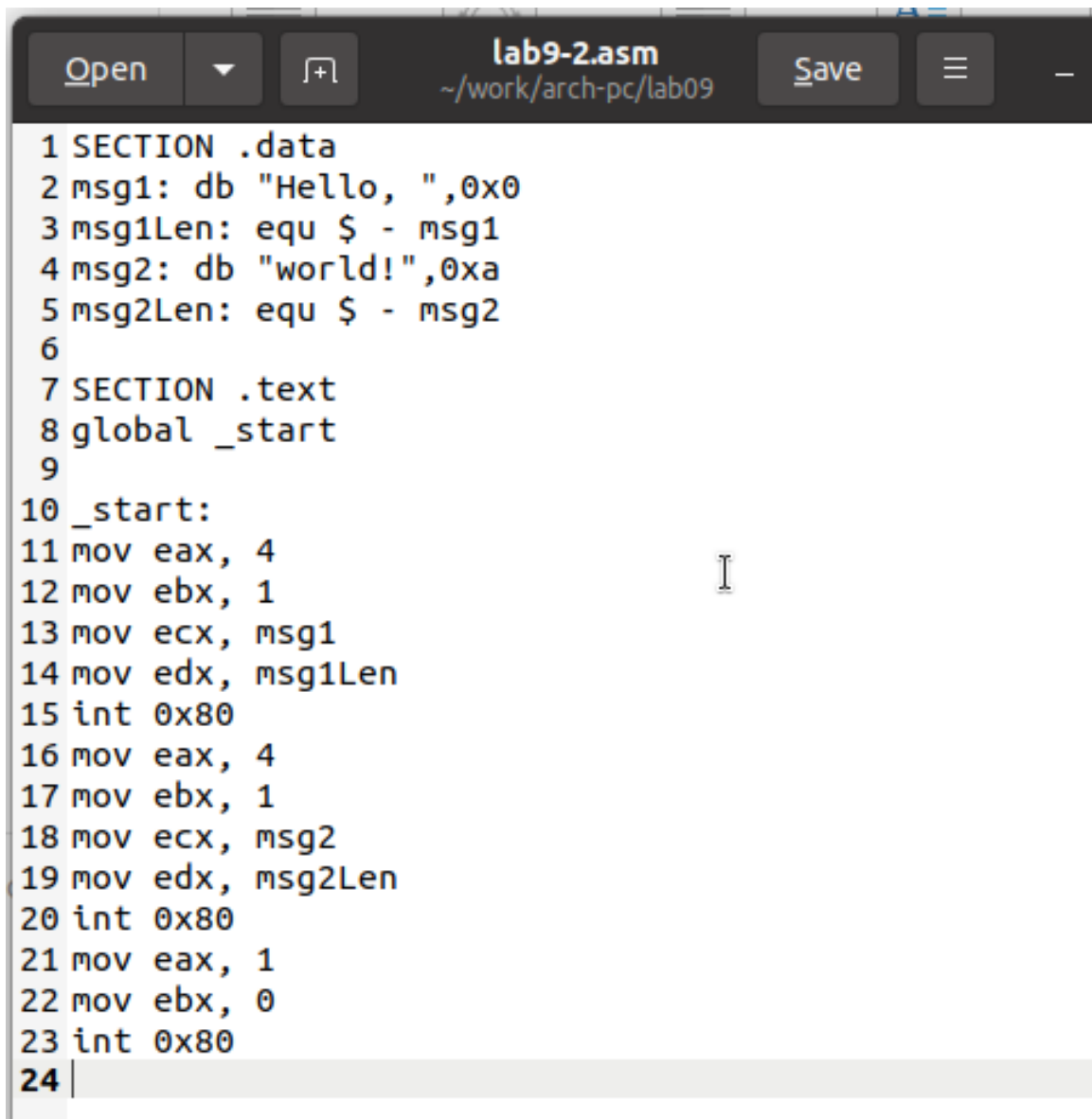
Рис. 2.3: Программа в файле lab9-1.asm



```
gulamova@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
gulamova@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
gulamova@ubuntu:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 6
2(3x-1)+7=41
gulamova@ubuntu:~/work/arch-pc/lab09$
```

Рис. 2.4: Запуск программы lab9-1.asm

4. Я создала файл lab9-2.asm, в который вписала код программы из Листинга 9.2, который выводит на экран сообщение “Hello world!”.



```
1 SECTION .data
2 msg1: db "Hello, ",0x0
3 msg1Len: equ $ - msg1
4 msg2: db "world!",0xa
5 msg2Len: equ $ - msg2
6
7 SECTION .text
8 global _start
9
10 _start:
11 mov eax, 4
12 mov ebx, 1
13 mov ecx, msg1
14 mov edx, msg1Len
15 int 0x80
16 mov eax, 4
17 mov ebx, 1
18 mov ecx, msg2
19 mov edx, msg2Len
20 int 0x80
21 mov eax, 1
22 mov ebx, 0
23 int 0x80
24
```

Рис. 2.5: Программа в файле lab9-2.asm

После этого я получила исполняемый файл. Чтобы использовать отладчик GDB, мне нужно было добавить в исполняемый файл отладочную информацию. Для этого я скомпилировала программу с ключом '-g'. Затем я загрузила исполняемый файл в отладчик gdb и проверила, как работает программа, выполнив её в среде GDB с использованием команды run (или просто r).

```

gulamova@ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
gulamova@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
gulamova@ubuntu:~/work/arch-pc/lab09$
gulamova@ubuntu:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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:
<http://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) run
Starting program: /home/gulamova/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 19290) exited normally]
(gdb) █

```

Рис. 2.6: Запуск программы lab9-2.asm в отладчике

Чтобы более детально разобраться в программе, я поставила точку останова у метки `_start`, с которой начинается любая программа на ассемблере, и запустила её. Затем я взглянула на дизассемблированный код.

```
gulamova@ubuntu: ~/work/arch-pc/lab09

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(gdb) run
Starting program: /home/gulamova/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 19290) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000
(gdb) run
Starting program: /home/gulamova/work/arch-pc/lab09/lab9-2

Breakpoint 1, 0x08049000 in _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. 2.7: Дизассемблированный код

```
gulamova@ubuntu: ~/work/arch-pc/lab09

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) 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. 2.8: Дизассимилированный код в режиме интел

На предыдущем этапе я уже разместила брейкпоинт с именем `_start` и проверила это, используя команду `info breakpoints`, или просто `i b`. После этого я установила ещё одну точку останова на адрес определённой инструкции, который можно было найти в середине экрана, в левой колонке напротив соответствующей инструкции. Я выбрала адрес предпоследней инструкции (`mov ebx,0x0`) и поставила там брейкпоинт.

```
gulamova@ubuntu: ~/work/arch-pc/lab09

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 19294 In: _start L?? PC: 0x8049000
(gdb)
(gdb)
(gdb) b *0x8049031Breakpoint 2 at 0x8049031
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 <_start>
breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 <_start+49>
(gdb) 
```

Рис. 2.9: Точка остановки

Отладчик предоставляет возможность просмотра содержимого ячеек памяти и регистров, и при необходимости я могу вручную поменять значения регистров или переменных. Я выполнила пять инструкций с помощью команды `stepi` (или `si`) и наблюдала за тем, как меняются значения в регистрах.

```
gulamova@ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 < start+5>
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 19294 In: _start L?? PC: 0x8049005
eip      0x8049000 0x8049000 < _start>
eflags   0x202    [ IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) si
0x08049005 in _start ()
(gdb)
```

Рис. 2.10: Изменение регистров

```
gulamova@ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 < _start+22>
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 19294 In: _start L?? PC: 0x8049016
gs      0x0      0
(gdb) si
0x08049005 in _start ()
(gdb) si
0x0804900a in _start ()
(gdb) si
0x0804900f in _start ()
(gdb) si
0x08049014 in _start ()
(gdb) si
0x08049016 in _start ()
(gdb) █
```

Рис. 2.11: Изменение регистров

Я проверила значение переменной msg1 по её имени и значение переменной msg2, обратившись к ней по адресу.

Чтобы изменить значение регистра или ячейки памяти, я использовала команду set, указав ей имя регистра или адрес в качестве аргумента. Я изменила первый символ в переменной msg1.



```
gulamova@ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804901b 0x804901b <_start+27>
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 19294 In: _start L?? PC: 0x804901b
0x804901b in _start ()
(gdb)
(gdb) x/1sb &msg10x804a000 <msg1>:      "Hello, "
(gdb)
(gdb) x/1sb 0x804a0080x804a008 <msg2>:    "world!\n"
(gdb)
(gdb)
(gdb) x/1sb &msg10x804a000 <msg1>:      "hello, "
(gdb)
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "Lor!d!\n"
(gdb) 
```

Рис. 2.12: Изменение значения переменной

Также я вывела значение регистра `edx` в разных форматах: в шестнадцатеричном, в двоичном и в символьном.

```
gulamova@ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804901b 0x804901b <_start+27>
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 19294 In: _start L?? PC: 0x804901b
(gdb) p/t $eax$2 = 100
(gdb)
(gdb) p/s $ecx$3 = 134520832
(gdb)
(gdb) p/x $ecx$4 = 0x804a000
(gdb)
(gdb) p/s $edx$5 = 8
(gdb)
(gdb) p/t $edx$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb)
```

Рис. 2.13: Вывод значения регистра

И далее я изменила значение регистра `ebx`, воспользовавшись командой `set`.

```
gulamova@ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804901b 0x804901b <_start+27>
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 19294 In: _start L?? PC: 0x804901b
(gdb) p/s $edx$5 = 8
(gdb)
(gdb) p/t $edx$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb)
(gdb)
(gdb) p/s $ebx$8 = 50
(gdb)
(gdb) p/s $ebx
$9 = 2
(gdb)
```

Рис. 2.14: Вывод значения регистра

5. Я скопировала файл lab8-2.asm, который был создан в ходе выполнения восьмой лабораторной работы, содержащий программу для вывода аргументов командной строки на экран. Затем я сформировала из него исполняемый файл. Чтобы загрузить эту программу в отладчик gdb вместе с аргументами, мне понадобилось использовать ключ `-args`. После этого я успешно загрузила исполняемый файл в отладчик, не забыв указать необходимые аргументы.

Первым делом я установила точку останова до выполнения первой инструкции программы и запустила её.

Важно отметить, что адрес вершины стека находится в регистре `esp`, и именно по этому адресу расположено значение, показывающее количество аргументов командной строки, включая само имя программы. В моем случае, число аргументов составило пять: имя программы `lab9-3` и четыре аргумента - аргумент1, аргумент2 и 'аргумент 3'.

Я также исследовала другие значения в стеке: по адресу `[esp+4]` находится адрес в памяти, где расположено имя программы, по адресу `[esp+8]` - адрес первого аргумента, по адресу `[esp+12]` - второго, и так далее.

```
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:
<http://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 lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8
(gdb) run
Starting program: /home/gulamova/work/arch-pc/lab09/lab9-3 argument 1 argument 2 argument\ 3

Breakpoint 1, 0x080490e8 in _start ()
(gdb) x/x $esp
0xffffd1a0: 0x00000006
(gdb)
0xffffd1a4: 0xffffd36e
(gdb) x/s *(void**)($esp + 4)
0xffffd36e: "/home/gulamova/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd397: "argument"
(gdb) x/s *(void**)($esp + 12)
0xffffd3a0: "1"
(gdb) x/s *(void**)($esp + 16)
0xffffd3a2: "argument"
(gdb) x/s *(void**)($esp + 20)
0xffffd3ab: "2"
(gdb) x/s *(void**)($esp + 24)
0xffffd3ad: "argument 3"
(gdb)
```

Рис. 2.15: Вывод значения регистра

Шаг изменения адреса в стеке составляет 4 байта (`[esp+4]`, `[esp+8]`, `[esp+12]`).

Это связано с тем, что размер каждой переменной, хранящейся в стеке, равен четырем байтам.

6. Я модифицировала программу из восьмой лабораторной работы (первое задание для индивидуального выполнения), включив в нее подпрограмму для расчета функции  $f(x)$ .

```
lab9-4.asm
~/work/arch-pc/lab09

2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)= 5(2+x)',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintLF
11 pop ecx
12 pop edx
13 sub ecx,1
14 mov esi, 0
15
16 next:
17 cmp ecx,0h
18 jz _end
19 pop eax
20 call atoi
21 call _fx
22 add esi,eax
23
24 loop next
25
26 _end:
27 mov eax, msg
28 call sprint
29 mov eax, esi
30 call iprintLF
31 call quit
32
33 _fx:
34 mov ebx,5
35 add eax,2
36 mul ebx
37 ret
38
```

Рис. 2.16: Программа в файле lab9-4.asm

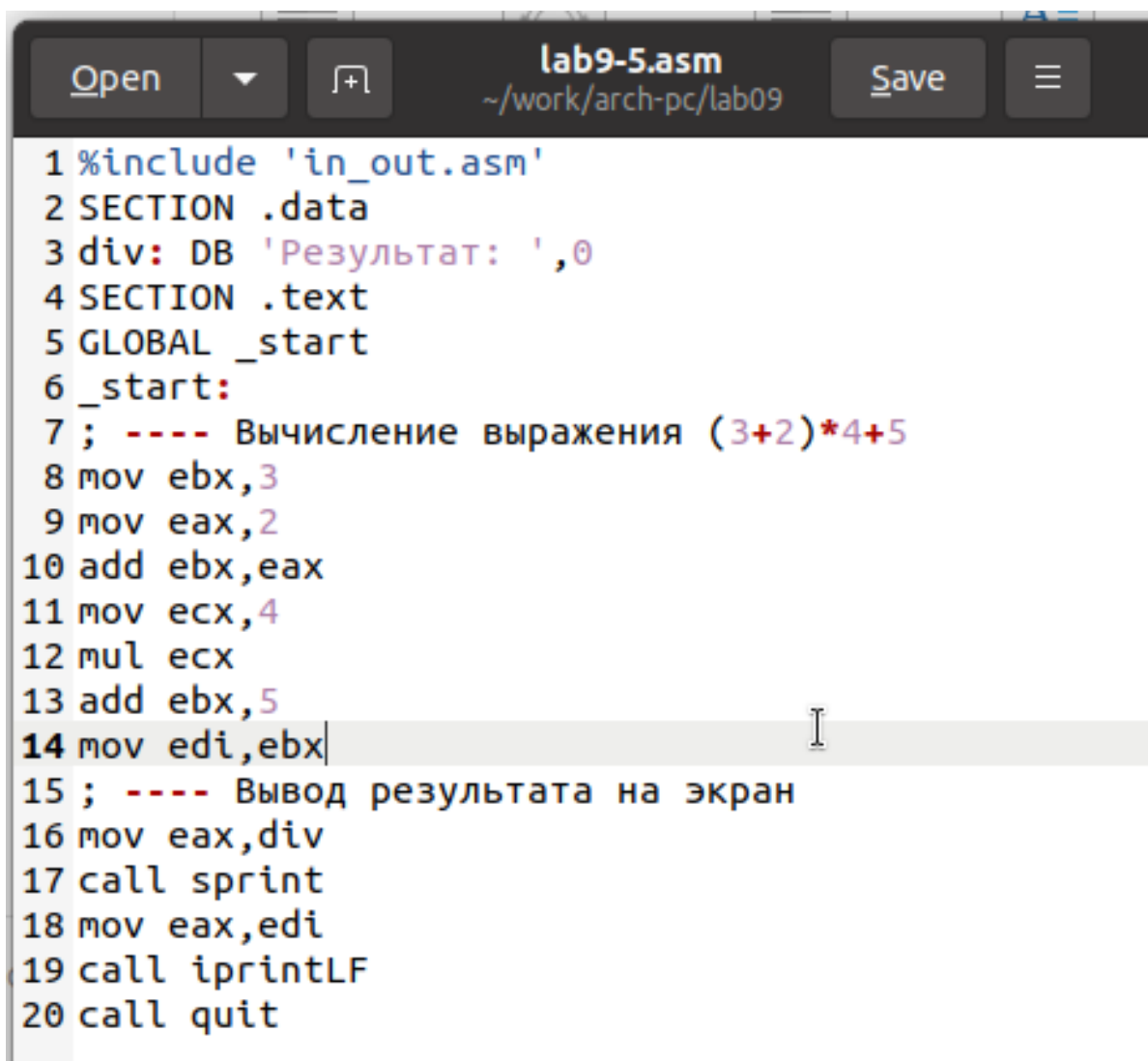
```

gulamova@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab9-4.asm
gulamova@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
gulamova@ubuntu:~/work/arch-pc/lab09$ ./lab9-4
f(x)= 5(2+x)
Результат: 0
gulamova@ubuntu:~/work/arch-pc/lab09$ ./lab9-4 0
f(x)= 5(2+x)
Результат: 10
gulamova@ubuntu:~/work/arch-pc/lab09$ ./lab9-4 0
f(x)= 5(2+x)
Результат: 10
gulamova@ubuntu:~/work/arch-pc/lab09$ ./lab9-4 0 3 4 6
f(x)= 5(2+x)
Результат: 105
gulamova@ubuntu:~/work/arch-pc/lab09$

```

Рис. 2.17: Запуск программы lab9-4.asm

7. В листинге приведена программа вычисления выражения  $(3 + 2) * 4 + 5$ . При запуске данная программа дает неверный результат. Проверил это. С помощью отладчика GDB, анализируя изменения значений регистров, определяю ошибку и исправлю ее.



```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,ebx
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
```

Рис. 2.18: Код с ошибкой

```
gulamova@ubuntu: ~/work/arch-pc/lab09
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa     10
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0xa     10
eip      0x8049100 0x8049100 <_start+24>
eflags   0x206    [ PF IF ]

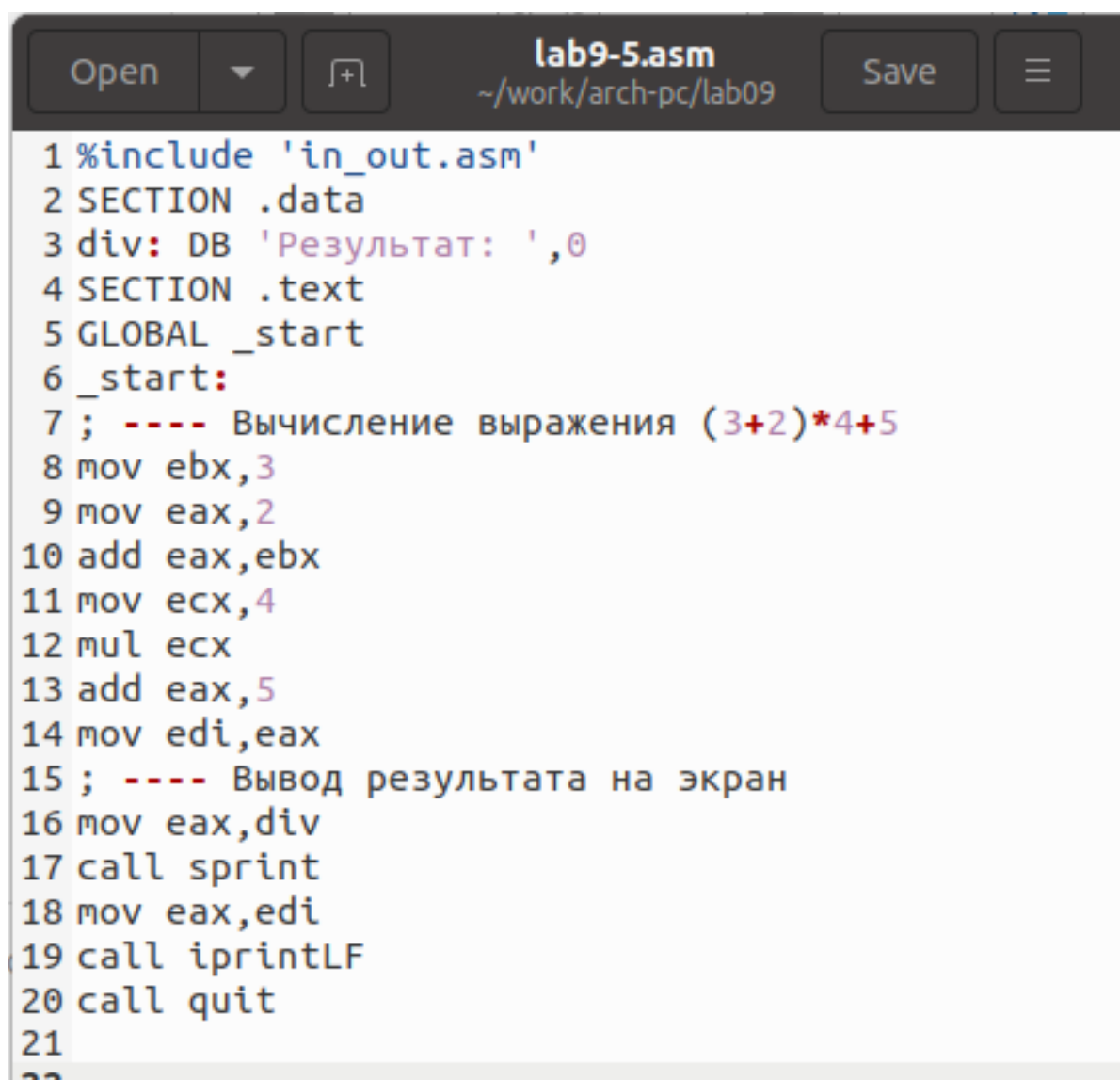
B+ 0x80490e8 <_start>    mov     ebx,0x3
B+ 0x80490e8 <_start+5>  mov     ebx,0x3
0x80490ed <_start+5>    mov     eax,0x2
0x80490f2 <_start+10>   add     ebx,eax
0x80490f4 <_start+12>   mov     ecx,0x4
0x80490f9 <_start+17>   mul     ecx,0x5
0x80490fb <_start+19>   add     ebx,0x5
>0x80490fe <_start+22>   mov     edi,ebx04a000
0x8049100 <_start+24>   mov     eax,0x804a000rint>
0x8049105 <_start+29>   call    0x804900f <sprint>
0x804910a <_start+34>   mov     eax,edi86 <iprintLF>
0x804910c <_start+36>   call    0x8049086 <iprintLF>

native process 19339 In: _start L?? PC: 0x8049100
(gdb) sNo process In: L?? PC: ??
0x080490f9 in _start ()
(gdb) si
0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) si
0x08049100 in _start ()
(gdb) c
Continuing.
Результат: 10
[Inferior 1 (process 19339) exited normally]
(gdb)
```

Рис. 2.19: Отладка

Я обнаружила, что в инструкции add были перепутаны местами аргументы, и после завершения работы значение регистра ebx ошибочно передавалось в edi вместо ожидаемого eax. Эту ошибку мне предстоит исправить.





```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add eax,ebx
11 mov ecx,4
12 mul ecx
13 add eax,5
14 mov edi,eax
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
21
22
```

Рис. 2.20: Код исправлен

```
gulamova@ubuntu: ~/work/arch-pc/lab09

eax      0x19      25
ecx      0x4       4
edx      0x0       0
ebx      0x3       3
esp      0xffffd1e0 0xffffd1e0
ebp      0x0       0
esi      0x0       0
edi      0x19      25
eip      0x8049100 0x8049100 <_start+24>
eflags   0x202     [ IF ]

B+ 0x80490e8 <_start>      mov     ebx,0x3
B+ 0x80490e8 <_start+5>    mov     ebx,0x3
0x80490ed <_start+5>      mov     eax,0x2
0x80490f2 <_start+10>     add     eax,ebx
0x80490f4 <_start+12>     mov     ecx,0x4
0x80490f9 <_start+17>     mul     ecx,0x5
0x80490fb <_start+19>     add     eax,0x5
>0x80490fe <_start+22>    mov     edi,eax04a000
0x8049100 <_start+24>    mov     eax,0x804a000rint>
0x8049105 <_start+29>    call   0x804900f <sprint>
0x804910a <_start+34>    mov     eax,edi86 <iprintLF>
0x804910c <_start+36>    call   0x8049086 <iprintLF>

native process 19351 In: _start L?? PC: 0x8049100
(gdb) sNo process In: L?? PC: ??
0x080490f9 in _start ()
(gdb) si
0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) si
0x08049100 in _start ()
(gdb) c
Continuing.
Результат: 25
[Inferior 1 (process 19351) exited normally]
(gdb) █
```

Рис. 2.21: Проверка работы

## 3 Выводы

Освоили работу с подпрограммами и отладчиком.