

Отчёта по лабораторной работе 10

Понятие подпрограммы. Отладчик GDB.

Негин Голчин Задех

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	28

Список иллюстраций

2.1	Файл lab10-1.asm	7
2.2	Работа программы lab10-1.asm	8
2.3	Файл lab10-1.asm	9
2.4	Работа программы lab10-1.asm	10
2.5	Файл lab10-2.asm	11
2.6	Работа программы lab10-2.asm в отладчике	12
2.7	дисассимилированный код	13
2.8	дисассимилированный код в режиме интел	14
2.9	точка остановки	15
2.10	изменение регистров	16
2.11	изменение регистров	17
2.12	изменение значения переменной	18
2.13	вывод значения регистра	19
2.14	вывод значения регистра	20
2.15	вывод значения регистра	21
2.16	Файл lab10-4.asm	22
2.17	Работа программы lab10-4.asm	23
2.18	код с ошибкой	24
2.19	отладка	25
2.20	код исправлен	26
2.21	проверка работы	27

Список таблиц

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

1. Создаем каталог для выполнения лабораторной работы № 10, переходим в него и создаем файл lab10-1.asm:
2. рассмотрим программу вычисления арифметического выражения $f(x) = 2x+7$ с помощью подпрограммы calcul. (Листинг 10.1). (рис. 2.1, 2.2)

```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0

SECTION .bss
x: RESB 80
rez: RESB 80

SECTION .text
GLOBAL _start
_start:
    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, [rez]
    call iprintLF
    call quit
_calcul:
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [rez], eax
    ret ; выход из подпрограммы
```

Рис. 2.1: Файл lab10-1.asm

```
[golchinzadeh@fedora lab10]$ nasm -f elf lab10-1.asm
[golchinzadeh@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[golchinzadeh@fedora lab10]$ ./lab10-1
Введите x: 6
2x+7=19
[golchinzadeh@fedora lab10]$ █
```

Рис. 2.2: Работа программы lab10-1.asm

3. Изменим текст программы, добавив подпрограмму subcalcul в подпрограмму calcul, для вычисления выражения $f(g(x))$, $f(x) = 2x + 7$, $g(x) = 3x - 1$ (рис. 2.3, 2.4)

```
x: RESB 80
rez: RESB 80

SECTION .text
GLOBAL _start
_start:
    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, [rez]
    call iprintLF
    call quit

_calcul:
    call _subcalcul
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [rez], eax
    ret ; выход из подпрограммы

_subcalcul:
    mov ebx, 3
    mul ebx
    sub eax, 1
    ret|
```

Рис. 2.3: Файл lab10-1.asm

```
[golchinzadeh@fedora lab10]$ nasm -f elf lab10-1.asm
[golchinzadeh@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[golchinzadeh@fedora lab10]$ ./lab10-1
Введите x: 6
2(3x-1)+7=41
```

Рис. 2.4: Работа программы lab10-1.asm

4. Создаем файл lab10-2.asm с текстом программы из Листинга 10.2. (Программа печати сообщения Hello world!): (рис. 2.5)

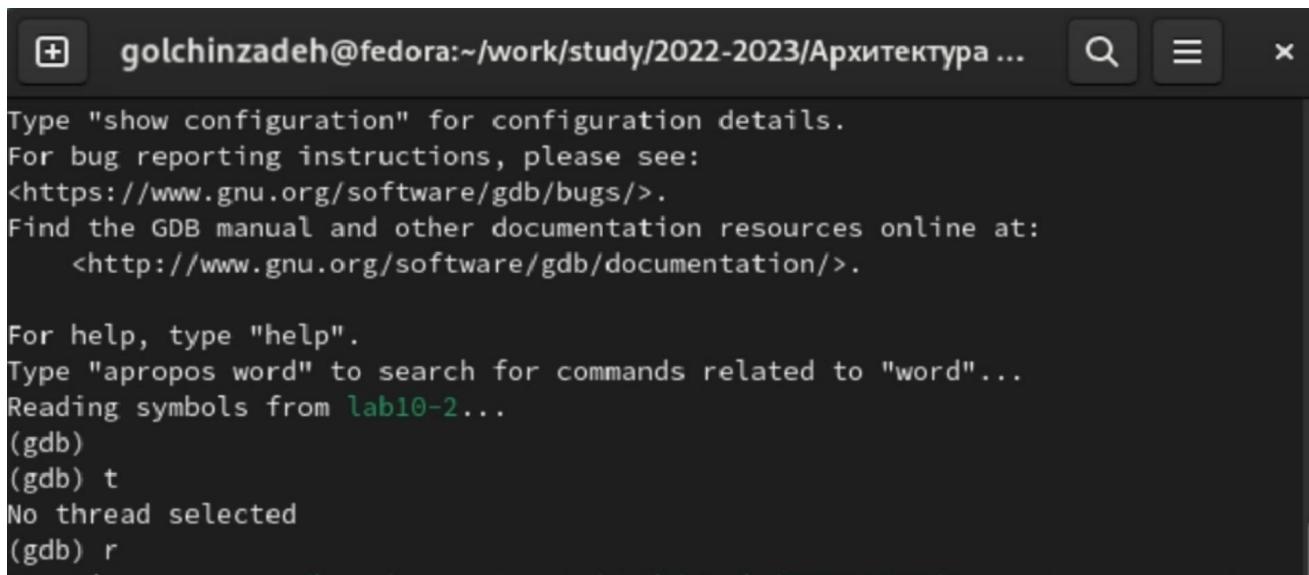
```
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1    [ ]
msg2: db "world!",0xa
msg2Len: equ $ - msg2

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. 2.5: Файл lab10-2.asm

Проверим работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r):(рис. 2.6)



golchinzadeh@fedora:~/work/study/2022-2023/Архитектура ...

```
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 lab10-2...
(gdb)
(gdb) t
No thread selected
(gdb) r
```

Рис. 2.6: Работа программы lab10-2.asm в отладчике

Для более подробного анализа программы установим брейкпоинт на метку `start` и запустим её. Посмотрите дисассимилированный код программы (рис. 2.7, 2.8)

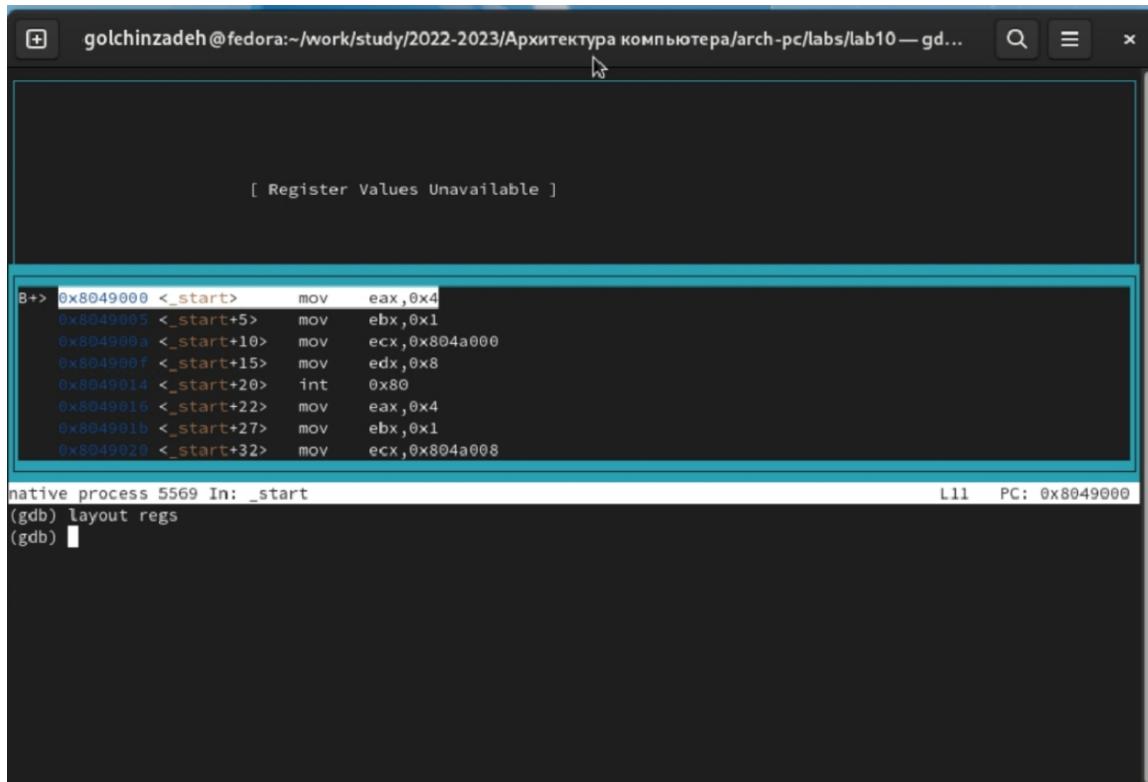
```
Breakpoint 1, _start () at lab10-2.asm:11
11      mov eax, 4
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:    mov    $0x4,%eax
  0x08049005 <+5>:    mov    $0x1,%ebx
  0x0804900a <+10>:   mov    $0x804a000,%ecx
  0x0804900f <+15>:   mov    $0x8,%edx
  0x08049014 <+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. 2.7: дисассимилированный код

```
golchinzadeh@fedora:~/work/study/2022-2023/Архитектура ...  
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: дисассимилированный код в режиме интел

Проверим метку с помощью команды `info breakpoints` (кратко `i b`) Установим еще одну точку останова по адресу инструкции. Определим адрес предпоследней инструкции (`mov ebx,0x0`) и установим точку.(рис. 2.9)



golchinzadeh@fedora:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10 — gd...

[Register Values Unavailable]

```
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
native process 5569 In: _start
(gdb) layout regs
(gdb) █
```

L11 PC: 0x8049000

Рис. 2.9: точка остановки

Выполните 5 инструкций с помощью команды stepi (или si) и проследим за изменением значений регистров. (рис. 2.11 2.12)

```

Register group: general
eax          0x4          4
ecx          0x0          0
edx          0x0          0
ebx          0x0          0
esp          0xfffffd130  0xfffffd130
ebp          0x0          0x0
esi          0x0          0
edi          0x0          0

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

native process 5569 In: _start
L12  PC: 0x8049005
ebp          0x0          0x0
esi          0x0          0
edi          0x0          0
eip          0x8049000  0x8049000 <_start>
eflags        0x202      [ IF ]
cs           0x23        35
ss           0x2b        43
--Type <RET> for more, q to quit, c to continue without paging--ds      0x2b      43
es           0x2b        43
fs           0x0          0
gs           0x0          0
(gdb) si
(gdb)

```

Рис. 2.10: изменение регистров

```

Register group: general
eax          0x4          4
ecx          0x804a008    134520840
edx          0x8          8
ebx          0x1          1
esp          0xfffffd130  0xfffffd130
ebp          0x0          0x0
esi          0x0          0
edi          0x0          0

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 5569 In: _start                                         L19
--Type <RET> for more, q to quit, c to continue without paging--ds      0x2b
es          0x2b          43
fs          0x0           0
gs          0x0           0
(gdb) si
(gdb)

```

Рис. 2.11: изменение регистров

Посмотрим значение переменной msg1 по имени Посмотрим значение переменной msg2 по адресу Изменим значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. Изменим первый символ переменной msg1 Заменим любой символ во второй переменной msg2. (рис. 2.12)

```
native process 5569 In: _start
(gdb) si
(gdb) si
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}0x804a008='L'
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "Lorlde!\n\034"
(gdb) █
```

Рис. 2.12: изменение значения переменной

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. С помощью команды set изменим значение регистра ebx:(рис. 2.13)

```
native process 5569 In: _start
(gdb) p/t $eax
$2 = 100
(gdb) p/s $ecx
$3 = 134520840
(gdb) p/x $ecx
$4 = 0x804a008
(gdb) p/s $edx
$5 = 8
(gdb) p/t $edx
$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb) █
```

Рис. 2.13: вывод значения регистра

С помощью команды set изменим значение регистра ebx:(рис. 2.14)

```
native process 5569 In: _start
(gdb) p/s $edx
$5 = 8
(gdb) p/t $edx
$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb) set $ebx='2'
(gdb) p/s $ebx
$8 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$9 = 2
(gdb) █
```

Рис. 2.14: вывод значения регистра

5. Скопируем файл lab9-2.asm. Создадим исполняемый файл. Загрузите исполняемый файл в отладчик, указав аргументы

Для начала установим точку останова перед первой инструкцией в программе и запустим ее.

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы): Как видно, число аргументов равно 5 – это имя программы lab10-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

Посмотрим остальные позиции стека – по адресу [esp+4], которые располагают адрес в памяти где находится имя программы, по адресу [esp+8] и храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д. (рис. 2.15)

```
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 lab10-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 5.
(gdb) run
```

```
This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n])
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

Breakpoint 1, _start () at lab10-3.asm:5
5      pop ecx ; Извлекаем из стека в `есх` количество
(gdb) x/x $esp
0xfffffd130: 0x00000001
(gdb) x/s *(void**)*($esp + 4)
0xfffffd2de: "/home/golchinzadeh/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10/lab10-3"
(gdb) █
```

([esp+4], [esp+8], [esp+12] - шаг равен размеру переменной - 4 байтам. # Самостоятельная работа Преобразуем программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. (рис. 2.16 2.17)

```
SECTION .text
global _start
_start:
    mov eax, fx
    call sprintLF
    pop ecx
    pop edx
    sub ecx,1
    mov esi, 0

    next:
    cmp ecx,0h
    jz _end
    pop eax
    call atoi
    call calc
    add esi, eax
    I

    loop next

    _end:
    mov eax, msg
    call sprint
    mov eax, esi
    call iprintf
    call quit

    calc:
    add eax,1
    mov ebx,7
    mul ebx
    ret
```

Рис. 2.16: Файл lab10-4.asm

```
[golchinzadeh@fedora lab10]$ nasm -f elf lab10-4.asm
[golchinzadeh@fedora lab10]$ ld -m elf_i386 -o lab10-4 lab10-4.o
[golchinzadeh@fedora lab10]$ ./lab10-4 1
f(x)=7(x+1)
Результат: 14
[golchinzadeh@fedora lab10]$ ./lab10-4 1 3 7 9 11 13 15
f(x)=7(x+1)
Результат: 462
```

Рис. 2.17: Работа программы lab10-4.asm

Сделаем программу вычисления выражения $(3+2)^*4+5$. При запуске данная программа дает неверный результат. Проверим это. С помощью отладчика GDB, анализируя изменения значений регистров, определим ошибку и исправьте ее.(рис. 2.18 2.19 2.20 2.21)

```
%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
```

II

Рис. 2.18: код с ошибкой

```

Register group: general
eax      0x8          8          ecx      0x4          4
edx      0x0          0          ebx      0x5          5
esp      0xfffffd130  0xfffffd130  ebp      0x0          0x0
esi      0x0          0          edi      0x0          0
eip      0x80490fb    0x80490fb <_start+1>  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    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
> 0x80490fb <_start+19> add    ebx,0x5
0x80490fe <_start+22>   mov    edi,ebx
0x8049100 <_start+24>   mov    eax,0x804a000
0x8049105 <_start+29>   call   0x804900f <sprint>
0x804910a <_start+34>   mov    eax,edi

native process 5729 In: _start
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

Breakpoint 1, _start () at lab10-5.asm:8
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si

```

Рис. 2.19: отладка

Перепутан порядок аргументов у инструкции add и по окончании работы в edi отправляется ebx вместо eax

```
%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
```

Рис. 2.20: код исправлен

```
0x8049000 <slen>      push  ebx
0x8049001 <slen+1>     mov    ebx, eax
0x8049003 <nextchar>   cmp    BYTE PTR [eax], 0x0
0x8049006 <nextchar+3>  je    0x804900b <finished>
0x8049008 <nextchar+5>  inc    eax
0x8049009 <nextchar+6>  jmp    0x8049003 <nextchar>
0x804900b <finished>    sub    eax, ebx
0x804900d <finished+2>  pop    ebx
0x804900e <finished+3>  ret
0x804900f <ssprint>    push   edx
0x8049010 <ssprint+1>   push   ecx
0x8049011 <ssprint+2>   push   ebx
0x8049012 <ssprint+3>   push   eax
0x8049013 <ssprint+4>   call   0x8049000 <slen>
0x8049018 <ssprint+9>   mov    edx, eax
0x804901a <ssprint+11>  pop    eax
0x804901b <ssprint+12>  mov    ecx, eax
0x804901d <ssprint+14>  mov    ebx, 0x1
0x8049022 <ssprint+19>  mov    eax, 0x4
```

native No process In:

L?? PC: ??

3 Выводы

Приобрели навыки написания программ с использованием подпрограмм. Познакомились с методами отладки при помощи GDB и его основными возможностями.