

Отчет по лабораторной работе №10

Смирнов Дмитрий Романович

Содержание

1	Цель работы	1
2	Выполнение лабораторной работы	1
3	Задания для самостоятельной работы:	12
4	Выводы.....	16

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

Создам каталог для выполнения лабораторной работы № 10 и перейду в него и о и создам файл lab10-1.asm. Запишу программу из листинга 10.1

```
[smirnovd_03_22@10 ~]$ mkdir work/arch-pc/lab10  
[smirnovd_03_22@10 ~]$ cd work/arch-pc/lab10  
[smirnovd_03_22@10 lab10]$ touch lab10-1.asm  
[smirnovd_03_22@10 lab10]$
```

Рис. 1: Рис1

```

%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
rezs: 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,[res]
call iprintLF
call quit
_calcul:
mov ebx,2
mul ebx
add eax,7
mov [rez],eax
ret ; выход из подпрограммы

```

Рис. 2: Рис2

Изменяю код и получаю результат

```

%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result1 DB '3x-1=',0
result2: DB '2x+7=',0

SECTION .bss
x: RESB 80
res1: RESB 80
res2: 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
mov eax,result1
call sprint
mov eax,[res1]
call iprintLF
mov eax,result2
call sprint
mov eax,[res2]
call iprintLF
call quit

_calcul:
call subcalcul
mov ebx,2
mul ebx
add eax,7
mov [res2],eax
ret ; выход из подпрограммы

subcalcul:
mov ebx,3
mul ebx
sub eax,1
mov [res1],eax
ret

```

Рис. 3: Puc3

```
Введите x: 2
3x-1=5
2x+7=17
[smirnovd_03_22@10 lab10]$
```

Рис. 4: Puc4

Создам файл lab10-2.asm с текстом программы из Листинга 10.2. Получу исполняемый файл. Для работы с GDB в исполняемый файл добавлю отладочную информацию. Загружу исполняемый файл в отладчик gdb Для более подробного анализа программы установлю брейкпоинт на метку _start Посмотрю дисассимилированный код программы с помощью команды disassemble начиная с метки _start Переключусь на отображение команд с Intel'овским синтаксисом

```
[smirnovd_03_22@10 lab10]$ touch lab10-2.asm
[smirnovd_03_22@10 lab10]$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
[smirnovd_03_22@10 lab10]$ ld -m elf_i386 -o lab10-2 lab10-2.o
[smirnovd_03_22@10 lab10]$ gdb lab10-2
GNU gdb (GDB) Fedora 11.2-3.fc36
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.htm
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 lab10-2...
```

Рис. 5: Puc5

```
(gdb) run
Starting program: /home/smirnovd_03_22/work/arch-pc/lab10/lab10-2

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.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 51648) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 9.
```

Рис. 6: Puc6

```

(gdb) run
Starting program: /home/smirnovd_03_22/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.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.
(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. 7: Puc7

Включите режим псевдографики для более удобного анализа программы Установлю еще одну точку останова по адресу инструкции. Посмотрите информацию о всех установленных точках останова



```
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
b+ 0x8049031 <_start+49> mov     ebx,0x0
      0x8049036 <_start+54> int      0x80
      0x8049038          add     BYTE PTR [eax],al
      0x804903a          add     BYTE PTR [eax],al
      0x804903c          add     BYTE PTR [eax],al
      0x804903e          add     BYTE PTR [eax],al
      0x8049040          add     BYTE PTR [eax],al
```

native process 51659 In: _start

(gdb) layout regs

(gdb) info breakpoints

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep y		0x08049000	lab10-2.asm:9
breakpoint already hit 1 time					

(gdb) b *0x8049031

Breakpoint 2 at 0x8049031: file lab10-2.asm, line 20.

(gdb) i b

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep y		0x08049000	lab10-2.asm:9
breakpoint already hit 1 time					
2	breakpoint	keep y		0x08049031	lab10-2.asm:20

(gdb) █

Рис. 8: Рис8

Посмотрите значение переменной msg1 по имени Изменяю первый символ переменной msg1 Заменяю любой символ во второй переменной msg2.

```
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
b+  0x8049031 <_start+49>   mov     ebx,0x0
      0x8049036 <_start+54>   int     0x80
      0x8049038             add     BYTE PTR [eax],al
      0x804903a             add     BYTE PTR [eax],al
      0x804903c             add     BYTE PTR [eax],al
      0x804903e             add     BYTE PTR [eax],al
      0x8049040             add     BYTE PTR [eax],al

native process 51659 In: _start
eip           0x8049000          0x8049000 <_start>
eflags        0x202             [ IF ]
cs            0x23              35
ss            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) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg2='l'
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "lorld!\n\034"
(gdb)
```

Рис. 9: Рис9

Выведу в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра `edx`.



smirnovd_03_22@10:~/work/arch-pc/lab10 — gdb lab10-2



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]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x0	0
gs	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
    0x8049025 <_start+37> mov     edx,0x7
    0x804902a <_start+42> int      0x80
    0x804902c <_start+44> mov     eax,0x1
b+ 0x8049031 <_start+49> mov     ebx,0x0
    0x8049036 <_start+54> int      0x80
    0x8049038          add     BYTE PTR [eax],al
    0x804903a          add     BYTE PTR [eax],al
    0x804903c          add     BYTE PTR [eax],al
    0x804903e          add     BYTE PTR [eax],al
```

native process 51659 In: _start

L14 PC: 0x8049016

cs	0x23	35
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x0	0
gs	0x0	0

```
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) p/s $edx
$2 = 8
(gdb) p/t &edx
No symbol "edx" in current context.
(gdb) p/t $edx
$3 = 1000
(gdb) p/x $edx
$4 = 0x8
(gdb)
```

Рис. 10: Рис10

С помощью команды set изменяю значение регистра ebx

```
smirnovd_03_22@10:~/work/arch-pc/lab10 — gdb lab10-2

eax      0x8      8
eax      0x1      1
ecx      0x804a008 134520840
edx      0x7      7
esp      0xffffffff 0xffffffff
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eip      0x8049031 0x8049031 <_start+49>
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       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
    0x8049016 <_start+22> mov     eax,0x4
    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
B > 0x8049031 <_start+49> mov     ebx,0x0
    0x8049038          add     BYTE PTR [eax],al
    0x804903a          add     BYTE PTR [eax],al
    0x804903c          add     BYTE PTR [eax],al
    0x804903e          add     BYTE PTR [eax],al

native process 51659 In: _start L14 PC: 0x8049016
(gdb) si 20 9031
(gdb) p/t &edx
No symbol "edx" in current context.
(gdb) p/t $edx
$3 = 1000
(gdb) p/x $edx
$4 = 0x8
$5 = 0x8
$6 = 0x8
(gdb) set $ebx='2'
(gdb) p/s $ebx
$7 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$8 = 2
(gdb) c
Continuing.
lorld!

Breakpoint 2, _start () at lab10-2.asm:20
(gdb) c
```

Рис. 11: Рис11

Скопирую файл lab9-2.asm, созданный при выполнении лабораторной работы №9
Создам исполняемый файл. Для начала установлю точку останова перед первой инструкцией в программе и запущу ее.

```
[smirnovd_03_22@i10 lab10]$ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
[smirnovd_03_22@i10 lab10]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
[smirnovd_03_22@i10 lab10]$ ld -m elf_i386 -o lab10-3 lab10-3.o
[smirnovd_03_22@i10 lab10]$ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora 11.2-3.fc36
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-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
Starting program: /home/smirnovd_03_22/work/arch-pc/lab10/lab10-3 аргумент1 аргумент 2 аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.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.

Breakpoint 1, _start () at lab10-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd1a0: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd350: "/home/smirnovd_03_22/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd380: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd392: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd3a3: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd3a5: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb) |
```

Рис. 12: Рис12

3 Задания для самостоятельной работы:

№1

```
;-----7(x + 1)---
```

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

```
SECTION .data
```

```
msg1 db "Функция: 7(x+1) "
```

```
msg2 db "Результат: ",0
```

```
SECTION .bss
```

```
res1: RESB 80
```

```
res2: RESB 80
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
pop ecx
```

```
pop edx
```

```
sub ecx,1
```

```
mov esi, 0
```

```
next:
```

```
cmp ecx,0h
```

```
pop eax
```

```
call atoi
```

```
call qwerty
```

```
loop next
```

```
mov eax, msg1
```

```
call sprint
```

```
mov eax, msg2
```

```
call sprint
```

```
mov eax, esi
```

```
call iprintLF
```

```
call quit
```

```
qwerty:
```

```
add eax,1
```

```
mov ebx,7
```

```
mul ebx
```

```
add esi,eax
```

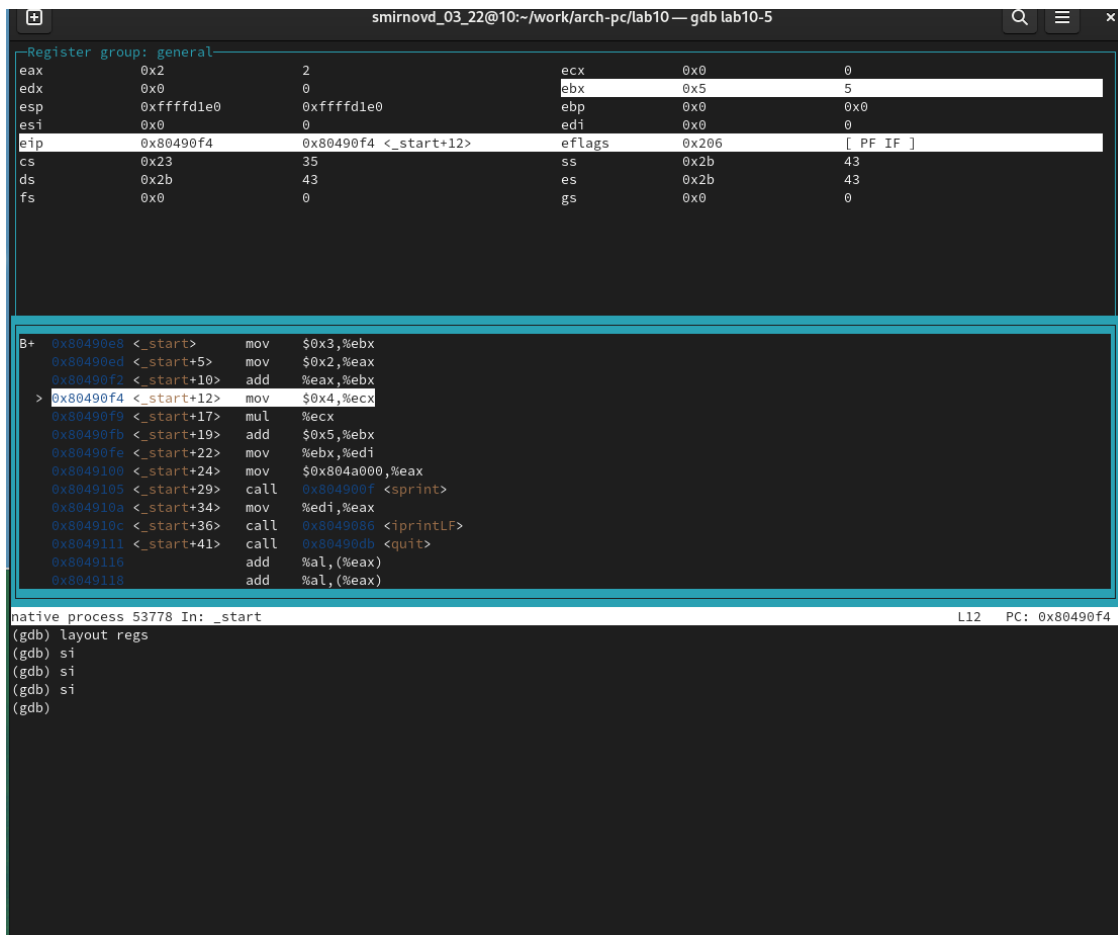
```
ret
```

Puc. 13: Puc13

```
[smirnovd_03_22@10 lab10]$ ./lab10-4 1 2 3 4
Функция: 7(x+1) Результат: Результат: 98
[smirnovd_03_22@10 lab10]$
```

Puc. 14: Puc14

№2



The screenshot shows a GDB debugger window titled "smirnovd_03_22@10:~/work/arch-pc/lab10 — gdb lab10-5". The window is divided into two main sections. The top section, titled "Register group: general", displays the values of various CPU registers. The bottom section shows the assembly code being executed, with the current instruction highlighted.

Register	Value	Register	Value
eax	0x2	ecx	0x0
edx	0x0	ebx	0x5
esp	0xfffffd1e0	ebp	0x0
esi	0x0	edi	0x0
eip	0x80490f4	eflags	0x206 [PF IF]
cs	0x23	ss	0x2b
ds	0x2b	es	0x2b
fs	0x0	gs	0x0

Address	Disassembly
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 \$0x804a000,%eax
0x8049105 <_start+29>	call 0x804900f <sprint>
0x804910a <_start+34>	mov %edi,%eax
0x804910c <_start+36>	call 0x8049086 <iprintLF>
0x8049111 <_start+41>	call 0x80490db <quit>
0x8049116	add %al,(%eax)
0x8049118	add %al,(%eax)

native process 53778 In: _start L12 PC: 0x80490f4

(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb)

Puc. 15: Puc15

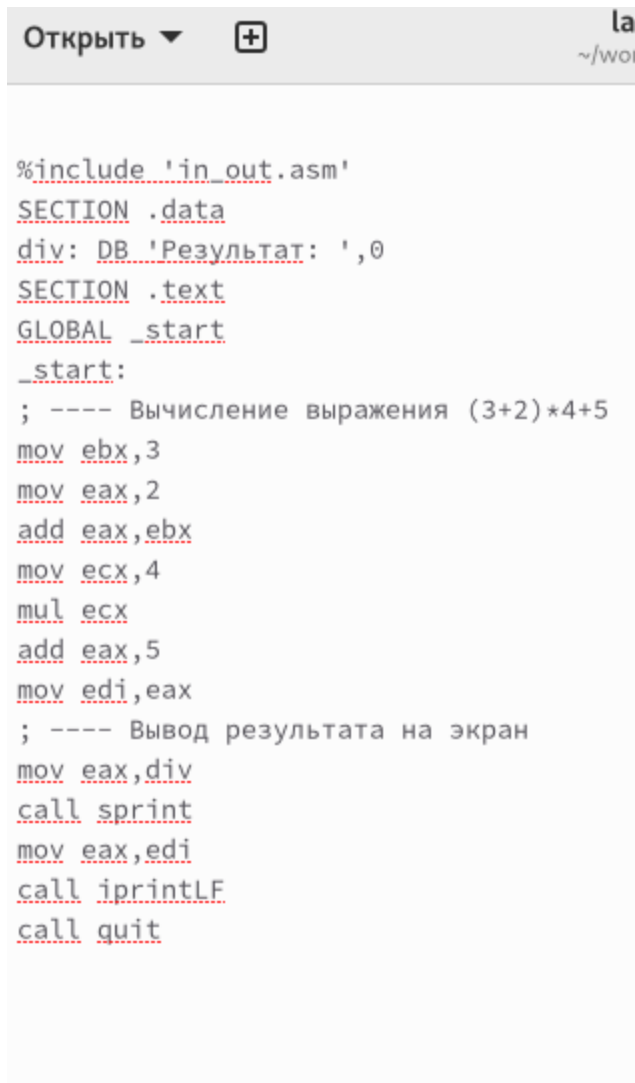
```
smirnovd_03_22@10:~/work/arch-pc/lab10 — gdb lab10-5

Register group: general
eax      0x8      8      ecx      0x4      4
edx      0x0      0      ebx      0x5      5
esp      0xffffd1e0 0xffffd1e0  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
0x80490f6 <_start+17>   mul     %ecx
> 0x80490fb <_start+19> add     $0x5,%ebx
0x80490fe <_start+22>   mov     %ebx,%edi
0x8049100 <_start+24>   mov     $0x804a000,%eax
0x8049105 <_start+29>   call    0x804900f <sprint>
0x804910a <_start+34>   mov     %edi,%eax
0x804910c <_start+36>   call    0x8049086 <iprintLF>
0x8049111 <_start+41>   call    0x80490db <quit>
0x8049116             add     %al,(%eax)
0x8049118             add     %al,(%eax)

native process 53778 In: _start L14 PC: 0x
(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) 
```

Puc. 16: Puc16



```
Открыть ▾ + la
~/woi

%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. 17: Рис17

4 Выводы

Я приобрел навык написания программ с использованием подпрограмм. Я познакомился с методами отладки при помощи GDB и его основными возможностями