

Отчёт по лабораторной работе 9

Архитектура компьютера

Тумуреева Галина Аркадьевна

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	8
2.3	Программа в файле lab9-1.asm	9
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

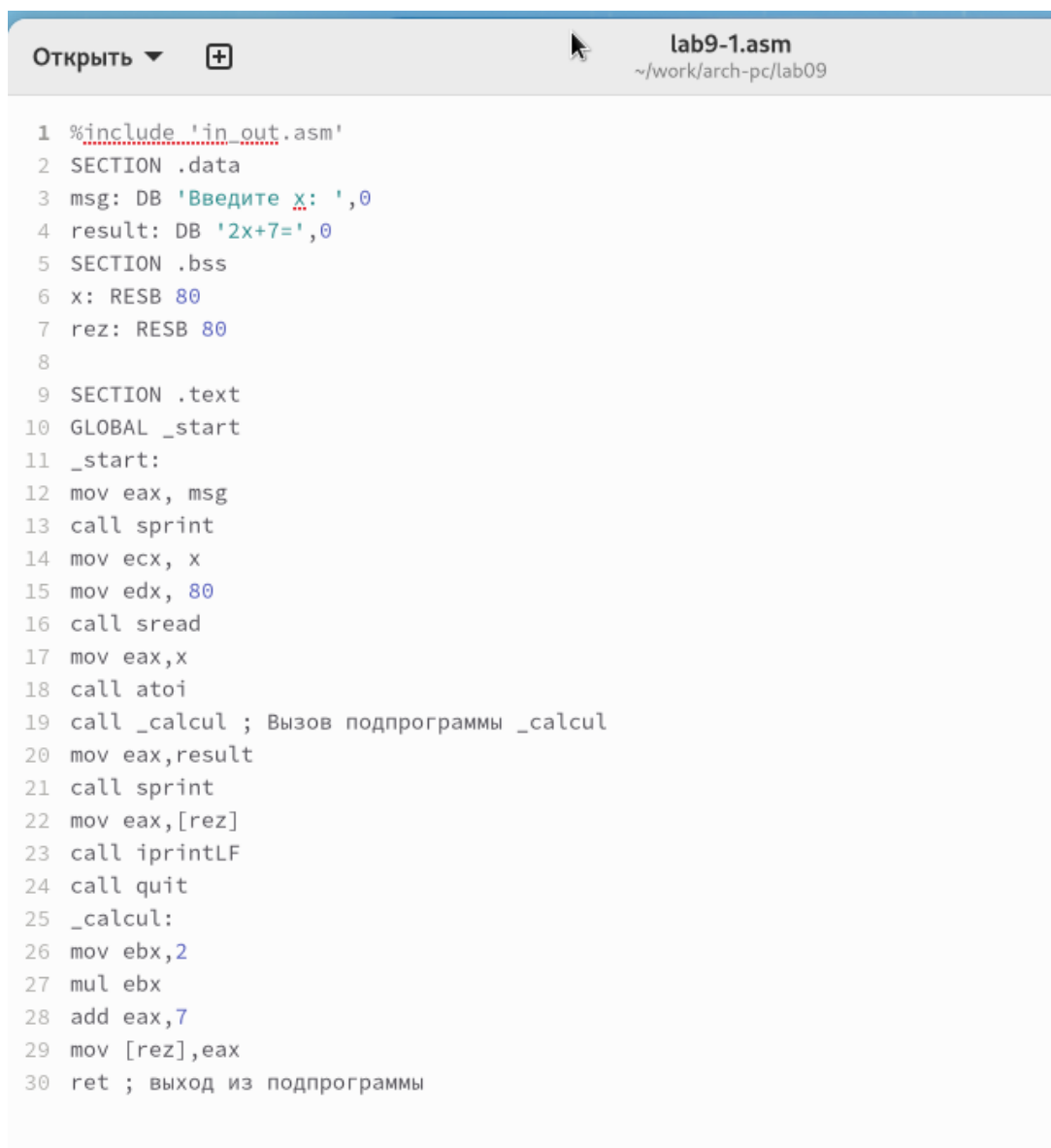
Список таблиц

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.



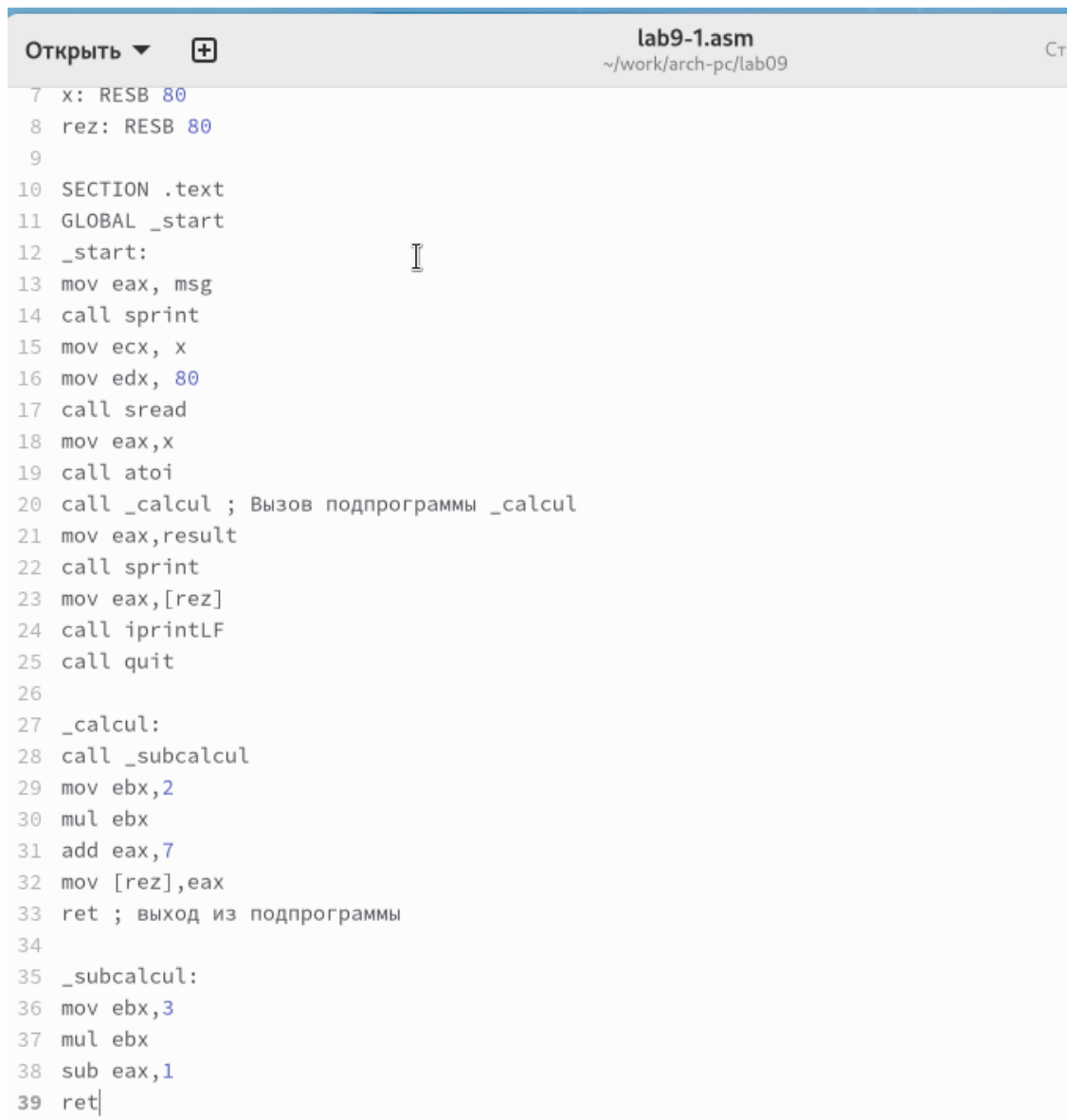
```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 rez: RESB 80
8
9 SECTION .text
10 GLOBAL _start
11 _start:
12 mov eax, msg
13 call sprint
14 mov ecx, x
15 mov edx, 80
16 call sread
17 mov eax, x
18 call atoi
19 call _calcul ; 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

```
[gtumureeva@gtumureeva lab09]$ nasm -f elf lab9-1.asm
[gtumureeva@gtumureeva lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[gtumureeva@gtumureeva lab09]$ ./lab9-1
Введите x: 4
2x+7=15
[gtumureeva@gtumureeva 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 $g(x) = 3x - 1$.




```
Открыть ▾  lab9-1.asm ~/work/arch-pc/lab09 Ст
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
```

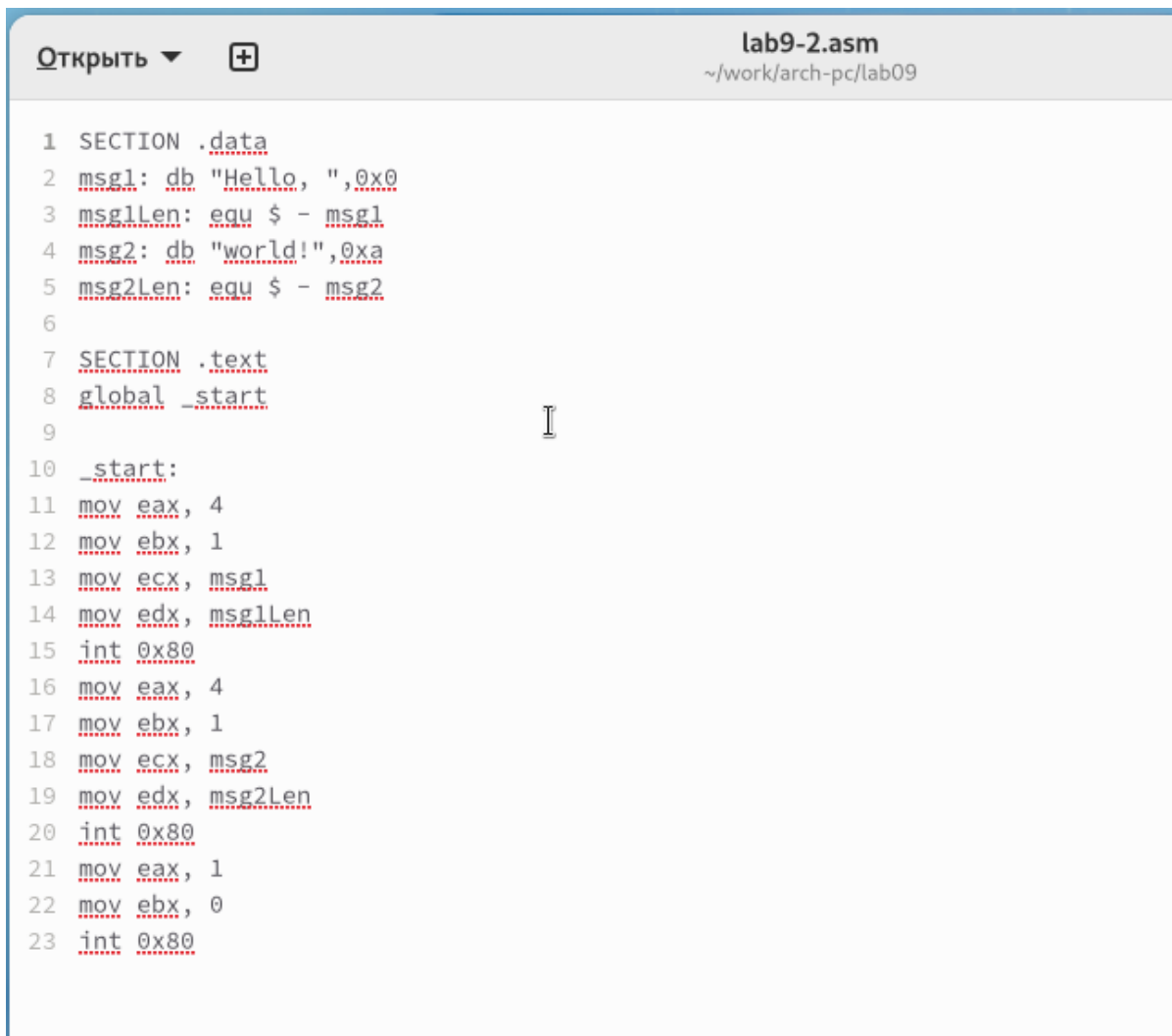
Рис. 2.3: Программа в файле lab9-1.asm



```
[gtumureeva@gtumureeva lab09]$
[gtumureeva@gtumureeva lab09]$ nasm -f elf lab9-1.asm
[gtumureeva@gtumureeva lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[gtumureeva@gtumureeva lab09]$ ./lab9-1
Введите x: 4
2(3x-1)+7=29
[gtumureeva@gtumureeva 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
```

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).

```

[gtumureeva@gtumureeva lab09]$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
[gtumureeva@gtumureeva lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[gtumureeva@gtumureeva lab09]$ gdb lab9-2
GNU gdb (GDB) Fedora 12.1-2.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 lab9-2...
(gdb) r
Starting program: /home/gtumureeva/work/arch-pc/lab09/lab9-2

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.
Hello, world!
[Inferior 1 (process 6497) 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.

```
gtumureeva@gtumureeva:~/work/arch-pc/lab09 — gdb lab9-2
Starting program: /home/gtumureeva/work/arch-pc/lab09/lab9-2

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.
Hello, world!
[Inferior 1 (process 6497) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 11.
(gdb) r
Starting program: /home/gtumureeva/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.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) 
```

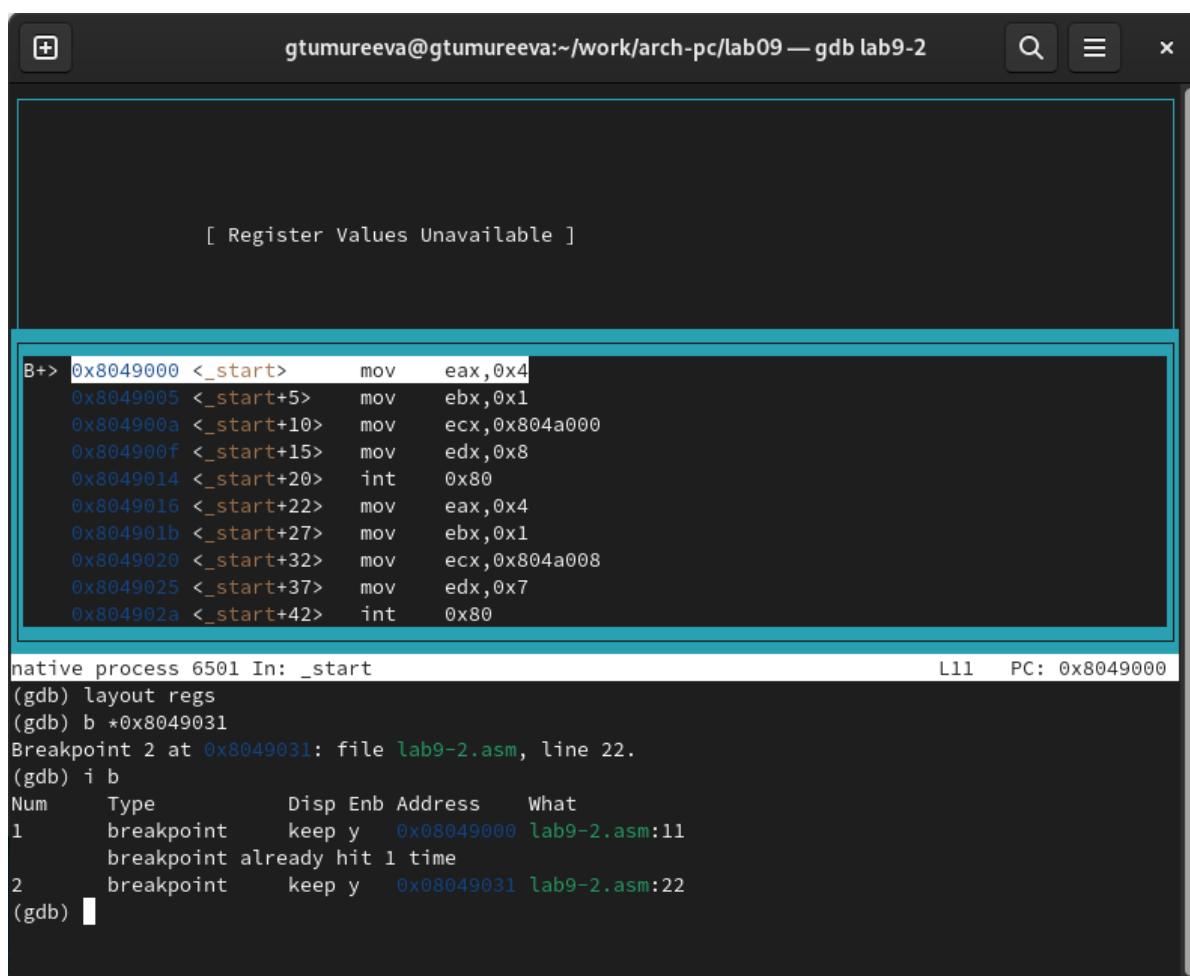
Рис. 2.7: Дизассимилированный код

```
gtumureeva@gtumureeva:~/work/arch-pc/lab09 — gdb lab9-2

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. 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.



The screenshot shows a GDB terminal window with the title bar "gtumureeva@gtumureeva:~/work/arch-pc/lab09 — gdb lab9-2". The main display area shows "[Register Values Unavailable]". Below this, a list of assembly instructions is displayed, starting with "B+> 0x8049000 <_start>". The instructions are:
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
Below the assembly list, the status bar shows "native process 6501 In: _start L11 PC: 0x8049000". The command history shows:
(gdb) layout regs
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 22.
(gdb) i b
A table of breakpoints is shown:

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep y		0x08049000	lab9-2.asm:11
	breakpoint already hit 1 time				
2	breakpoint	keep y		0x08049031	lab9-2.asm:22

(gdb)
The prompt "(gdb)" is followed by a cursor.

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.

```
gtumureeva@gtumureeva:~/work/arch-pc/lab09 — gdb lab9-2

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd210 0xffffd210
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
0x8049025 <_start+37> mov edx,0x7
0x804902a <_start+42> int 0x80

native process 6501 In: _start L12 PC: 0x8049005
eip      0x8049000 0x8049000 <_start>
eflags   0x202 [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--cs 0x23
35
ss       0x2b      43
ds       0x2b      43
es       0x2b      43
fs       0x0      0
gs       0x0      0
(gdb) si
(gdb)
```

Рис. 2.10: Изменение регистров

The screenshot shows a GDB terminal window with the title bar "gtumureeva@gtumureeva:~/work/arch-pc/lab09 — gdb lab9-2". The window is divided into three main sections. The top section, titled "Register group: general", displays the current values of the general-purpose registers:

Register	Value (hex)	Value (decimal)
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xffffd210	0xffffd210
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0

. The middle section displays assembly code starting from address 0x8049000, with the instruction at 0x8049016, `mov eax, 0x4`, highlighted. The bottom section shows the state of the native process 6501, including segment registers (ss, ds, es, fs, gs) and the current instruction pointer (PC) at 0x8049016. The GDB prompt "(gdb)" is visible at the bottom.

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd210 0xffffd210
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
0x8049025 <_start+37>   mov     edx,0x7
0x804902a <_start+42>   int     0x80

native process 6501 In: _start L16 PC: 0x8049016
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) si
(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`.

The screenshot shows a GDB terminal window with the title bar "gtumureeva@gtumureeva:~/work/arch-pc/lab09 — gdb lab9-2". The window is divided into three main sections. The top section, titled "Register group: general", displays the values of general-purpose registers:

Register	Value (hex)	Value (decimal)
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xffffd210	0xffffd210
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0

. The middle section displays assembly code starting from address 0x8049000. The instruction at 0x8049016, "mov eax, 0x4", is highlighted with a blue selection bar. The bottom section shows the execution of a program, with the prompt "(gdb) x/1sb &msg1" and the output "Hello, ". Subsequent commands show memory at 0x804a008 containing "world!\n\034", a modification of msg1 to 'h', and another memory dump showing "hello, " and "Lorld!\n\034".

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd210 0xffffd210
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
    0x8049025 <_start+37> mov     edx,0x7
    0x804902a <_start+42> int     0x80

native process 6501 In: _start L16 PC: 0x8049016
(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>: "Lorld!\n\034"
(gdb) 
```

Рис. 2.12: Изменение значения переменной

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

```
gtumureeva@gtumureeva:~/work/arch-pc/lab09 — gdb lab9-2

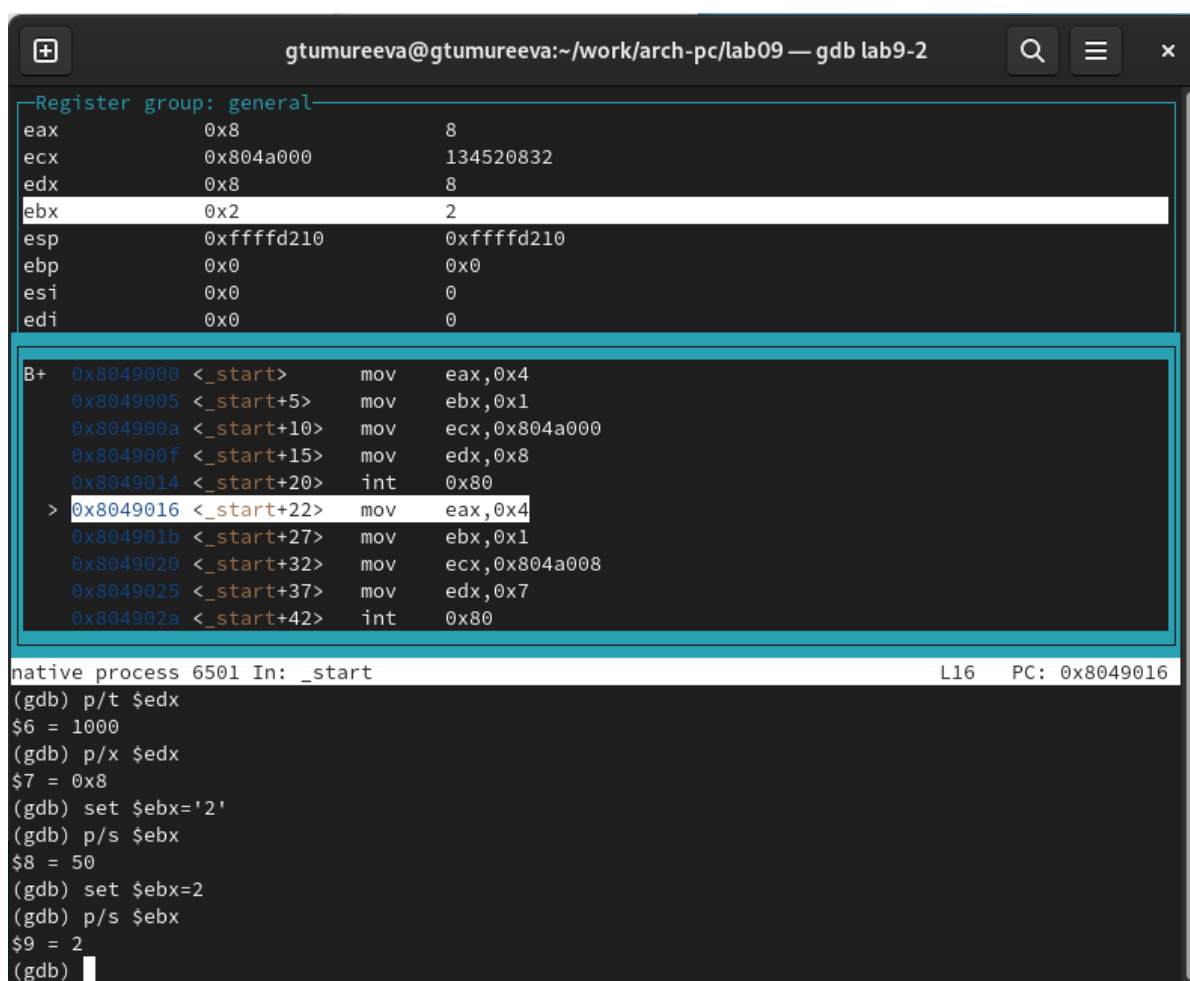
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd210 0xffffd210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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
   0x8049025 <_start+37> mov     edx,0x7
   0x804902a <_start+42> int      0x80

native process 6501 In: _start L16 PC: 0x8049016
(gdb) p/s $ecx
$3 = 134520832
(gdb) p/x $ecx
$4 = 0x804a000
(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

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



```
gtumureeva@gtumureeva:~/work/arch-pc/lab09 — gdb lab9-2

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd210 0xffffd210
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
0x8049025 <_start+37>   mov     edx,0x7
0x804902a <_start+42>   int     0x80

native process 6501 In: _start L16 PC: 0x8049016
(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) 
```

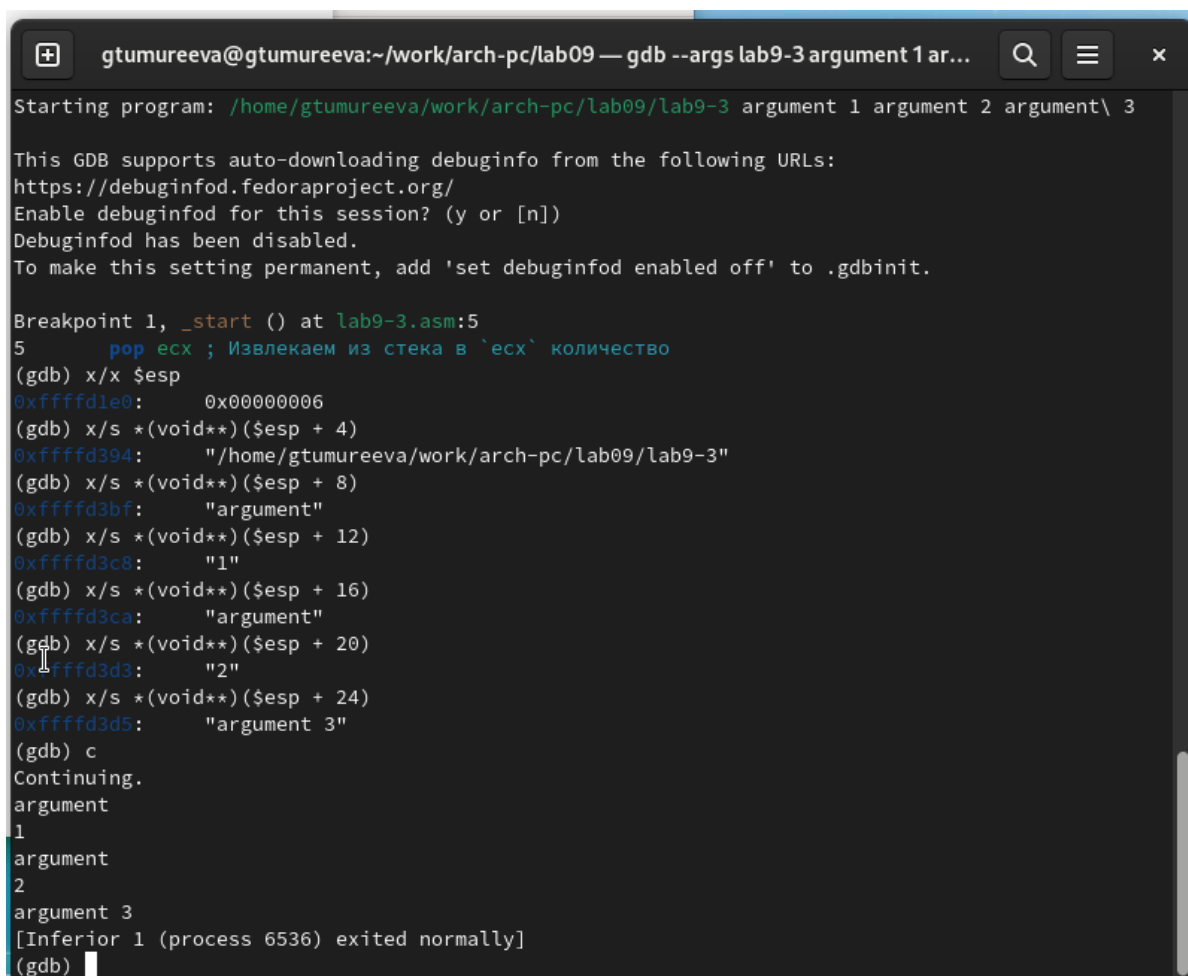
Рис. 2.14: Вывод значения регистра

5. 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.



```
gtumureeva@gtumureeva:~/work/arch-pc/lab09 — gdb --args lab9-3 argument 1 ar...
Starting program: /home/gtumureeva/work/arch-pc/lab09/lab9-3 argument 1 argument 2 argument\ 3

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.

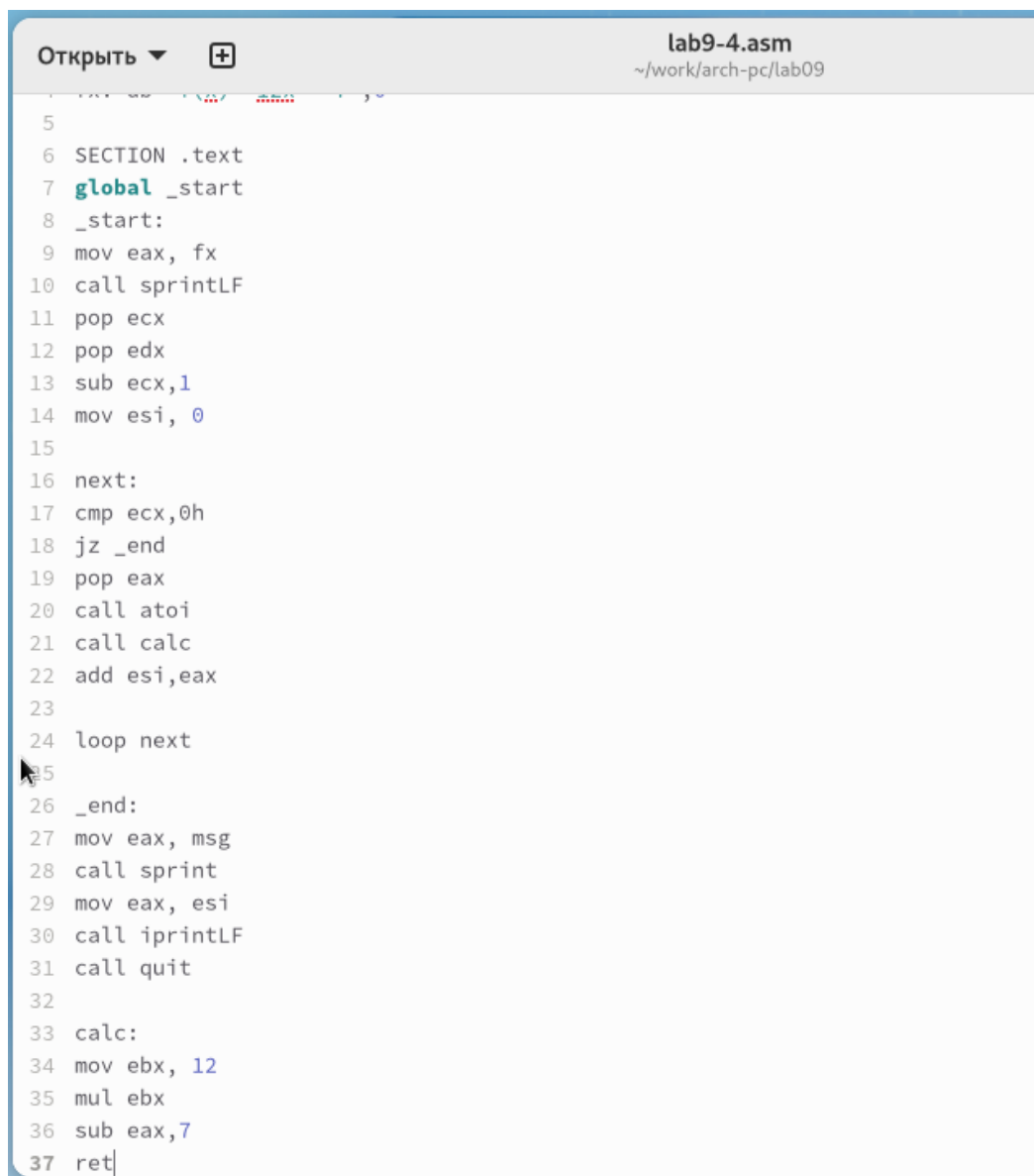
Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xffffd1e0:      0x00000006
(gdb) x/s *(void**)(esp + 4)
0xffffd394:      "/home/gtumureeva/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd3bf:      "argument"
(gdb) x/s *(void**)(esp + 12)
0xffffd3c8:      "1"
(gdb) x/s *(void**)(esp + 16)
0xffffd3ca:      "argument"
(gdb) x/s *(void**)(esp + 20)
0xffffd3d3:      "2"
(gdb) x/s *(void**)(esp + 24)
0xffffd3d5:      "argument 3"
(gdb) c
Continuing.
argument
1
argument
2
argument 3
[Inferior 1 (process 6536) exited normally]
(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 $f(x)$.




```
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 calc
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 calc:
34 mov ebx, 12
35 mul ebx
36 sub eax,7
37 ret
```

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

```
[gtumureeva@gtumureeva lab09]$ nasm -f elf lab9-4.asm
[gtumureeva@gtumureeva lab09]$ ld -m elf_i386 -o lab9-4 lab9-4.o
[gtumureeva@gtumureeva lab09]$ ./lab9-4 1
f(x)= 12x - 7
Результат: 5
[gtumureeva@gtumureeva lab09]$ ./lab9-4 1 3 5 4 6 3
f(x)= 12x - 7
Результат: 222
[gtumureeva@gtumureeva 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.

Открыть ▾ 

lab9-5.asm
~/work/arch-pc/lab09

```
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

The screenshot shows a GDB debugger window with the title bar "gtumureeva@gtumureeva:~/work/arch-pc/lab09 — gdb lab9-5". The window is divided into several panels. The top panel displays the state of CPU registers:

eax	0x8	8
ecx	0x4	4
edx	0x0	0
ebx	0xa	10
esp	0xffffd210	0xffffd210
ebp	0x0	0x0
esi	0x0	0
edi	0xa	10

. The middle panel shows assembly code with addresses and instructions:

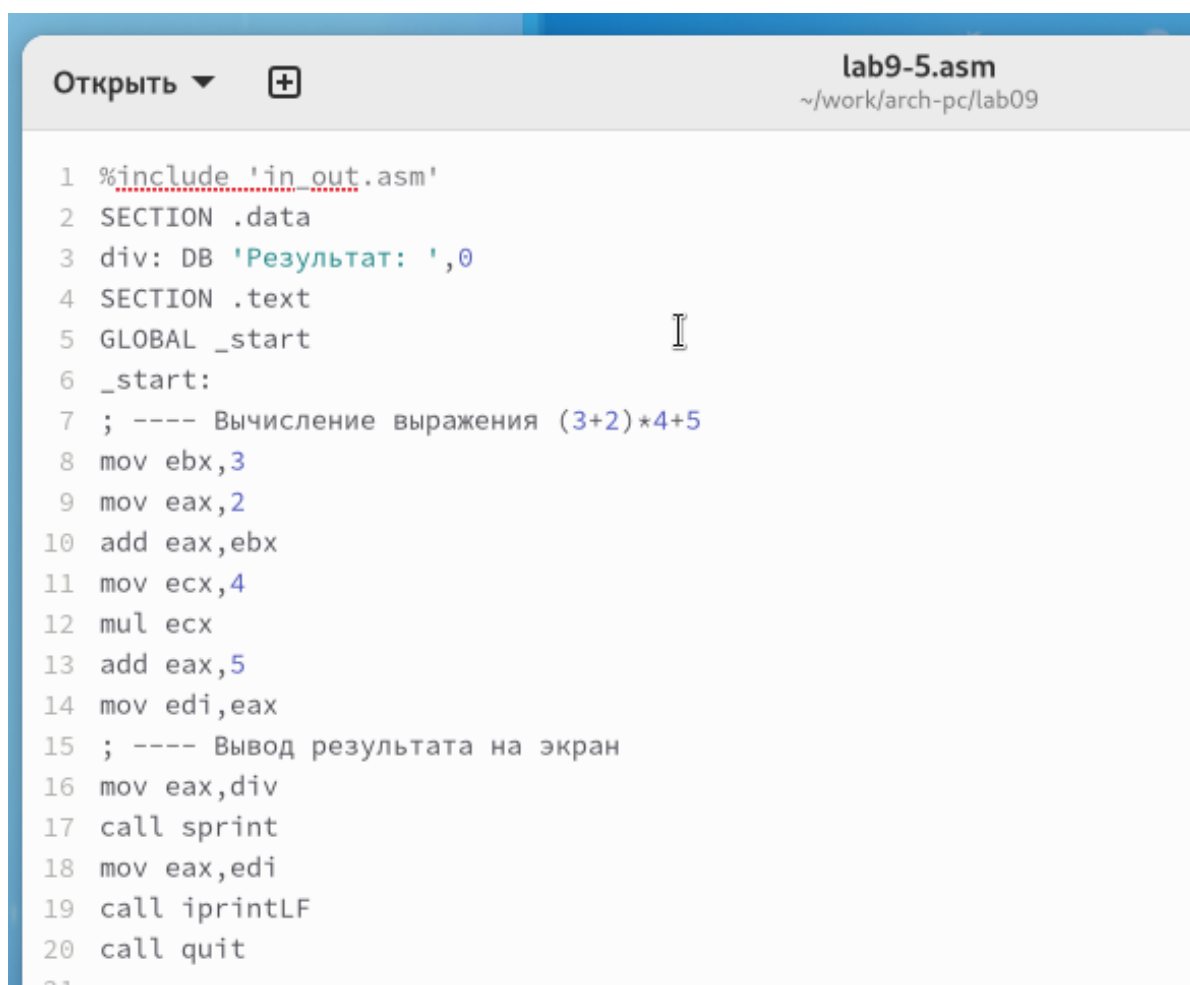
```
B+ 0x80490e8 <_start>    mov     ebx,0x3
0x8049100 <_start+24>  mov     eax,0x804a000
0x8049105 <_start+29>  call    0x804900f <sprint>
0x804910a <_start+34>  mov     eax,edi
0x804910c <_start+36>  call    0x8049086 <iprintLF>
0x8049111 <_start+41>  call    0x80490db <quit>
```

. Below the code, a value "04a000" is shown next to a label "rint>". The bottom panel shows the GDB command line with the following text:

```
native process 6655 In: _start L16 PC: 0x8049100
Breakpoint 0: No process In: 9-5.asm:8 L?? PC: ??
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) c
Continuing.
Результат: 10
[Inferior 1 (process 6655) 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.



The screenshot shows a code editor window titled "lab9-5.asm" with the path "~/work/arch-pc/lab09". The code is as follows:

```
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
```

Рис. 2.20: Код исправлен

```
gtumureeva@gtumureeva:~/work/arch-pc/lab09 — gdb lab9-5
eax      0x19      25
ecx      0x4       4
edx      0x0       0
ebx      0x3       3
esp      0xffffd210 0xffffd210
ebp      0x0       0
esi      0x0       0
edi      0x19      25

B+ 0x80490e8 <_start>    mov     ebx,0x3
0x8049100 <_start+24>    mov     eax,0x804a000
0x8049105 <_start+29>    call   0x804900f <sprint>
0x804910a <_start+34>    mov     ecx,0di
0x804910c <_start+36>    call   0x8049086 <iprintLF>
0x8049111 <_start+41>    call   0x80490db <quit>

>                                04a000
                                rint>

native process 6690 In: _start L16 PC: 0x8049100
Breakpoint 1: No process In: 9-5.asm:8 L?? PC: ??
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) c
Continuing.
Результат: 25
[Inferior 1 (process 6690) exited normally]
(gdb) 
```

Рис. 2.21: Проверка работы

3 Выводы

Освоили работу с подпрограммами и отладчиком.