

Отчёт по лабораторной работе 9

Архитектура компьютера

Хулер Александрович Оюн

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
	2.1 Задание для самостоятельной работы	20
3	Выводы	26

Список иллюстраций

2.1	Код программы lab9-1.asm	7
2.2	Компиляция и запуск программы lab9-1.asm	7
2.3	Код программы lab9-1.asm	8
2.4	Компиляция и запуск программы lab9-1.asm	9
2.5	Код программы lab9-2.asm	10
2.6	Компиляция и запуск программы lab9-2.asm в отладчике	11
2.7	Дизассемблированный код	12
2.8	Дизассемблированный код в режиме интел	13
2.9	Точка останова	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	Код программы lab9-4.asm	21
2.17	Компиляция и запуск программы lab9-4.asm	21
2.18	Код программы lab9-5.asm с ошибкой	22
2.19	Отладка	23
2.20	Код программы lab9-5.asm исправлен	24
2.21	Проверка работы	25

Список таблиц

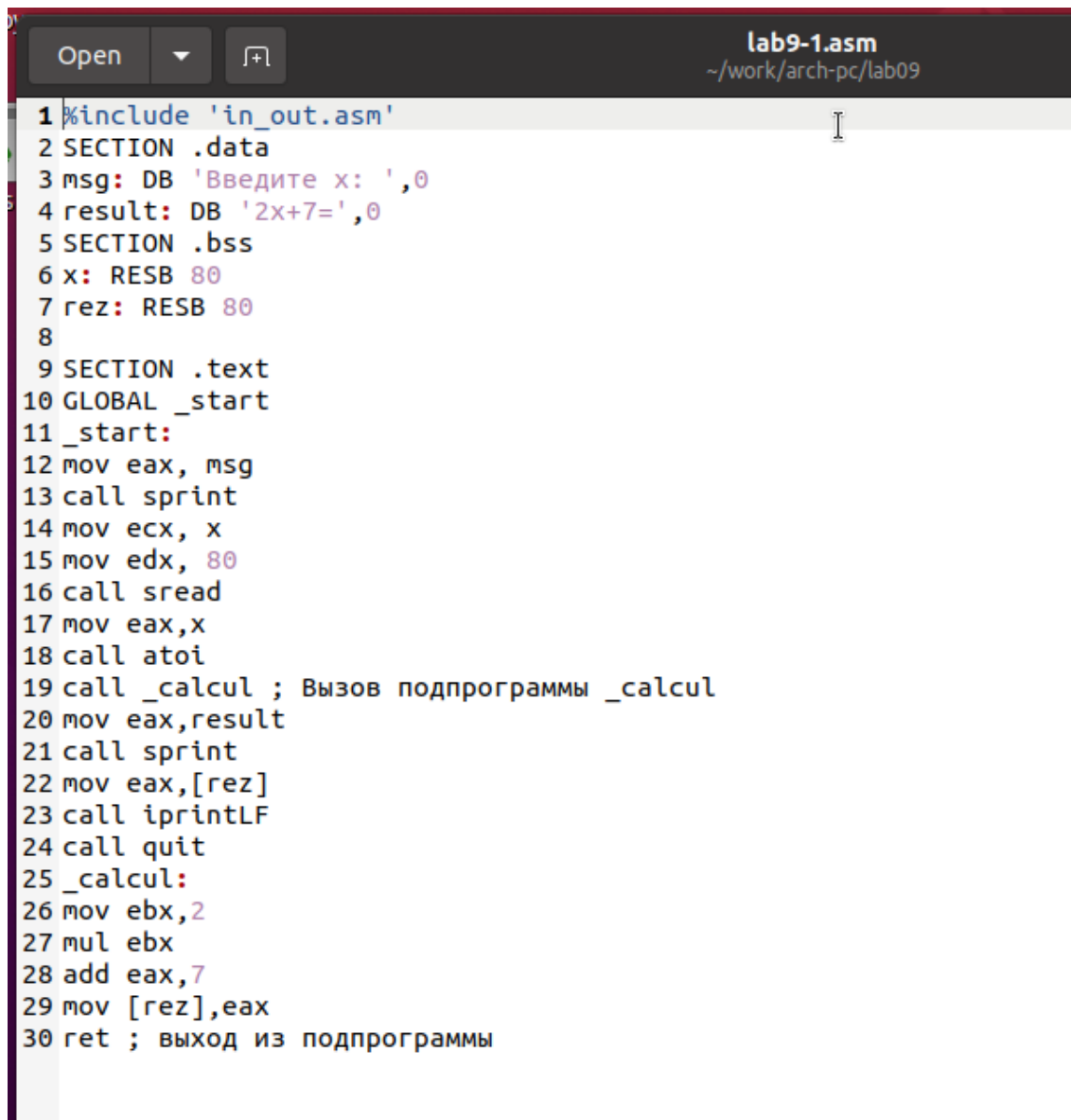
1 Цель работы

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

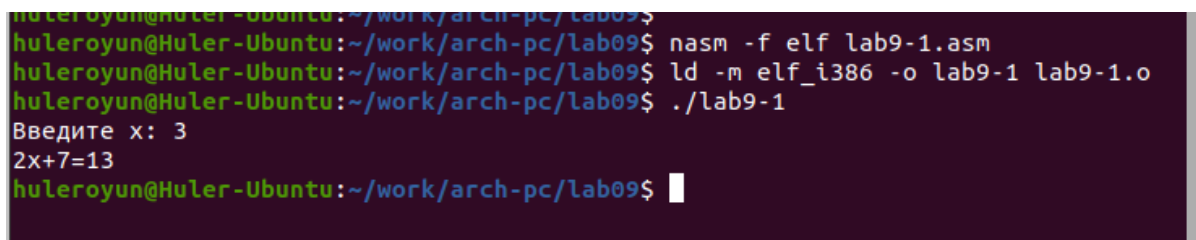
Я организовал папку для проведения лабораторного занятия № 9 и переместился в неё. После этого я создал файл с именем lab9-1.asm.

Давайте рассмотрим в качестве примера программу, задачей которой является расчёт арифметической формулы $f(x) = 2x + 7$, используя для этого вспомогательную функцию `calcul`. В этом случае значение x подаётся через клавиатуру, а расчёт формулы происходит внутри вспомогательной функции.



```
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
```

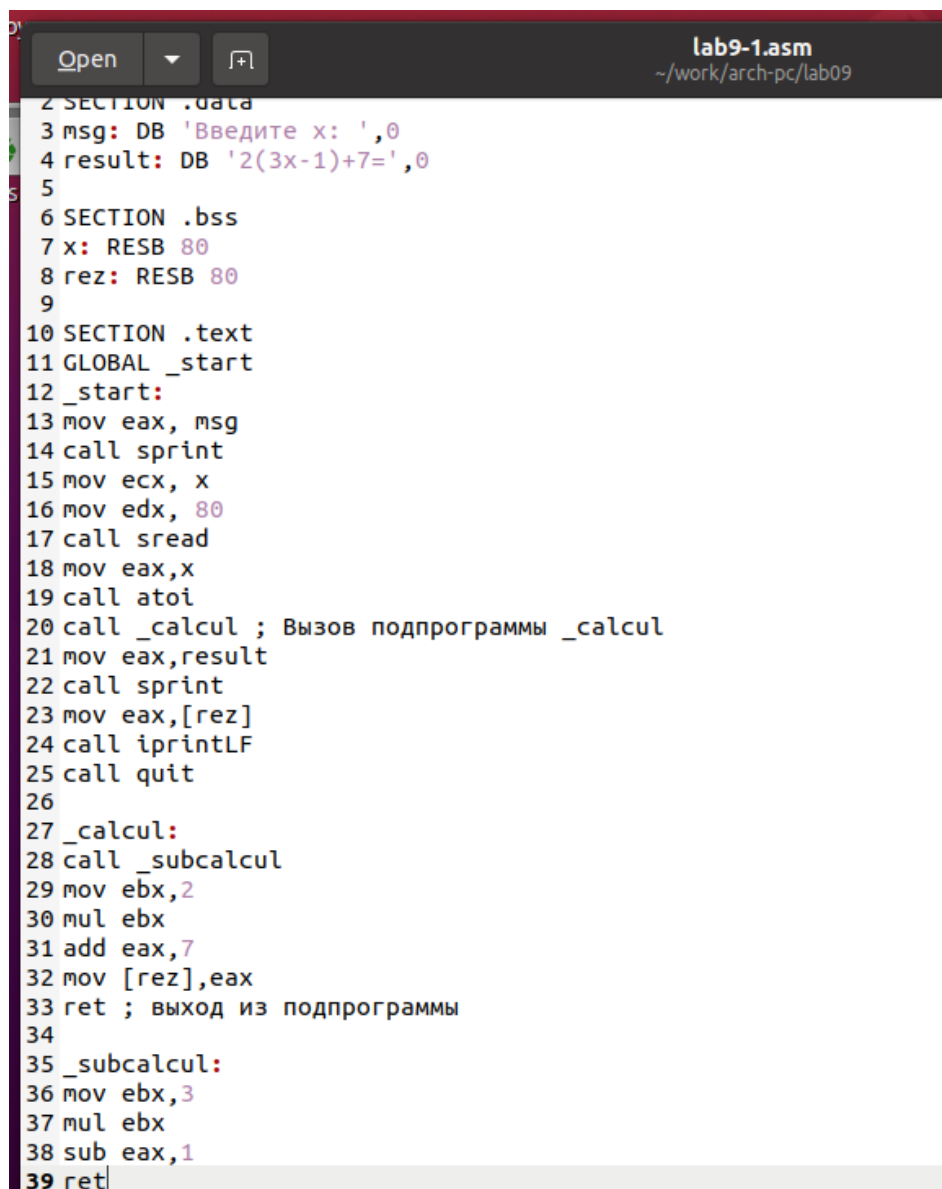
Рис. 2.1: Код программы lab9-1.asm



```
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab09$
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 3
2x+7=13
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab09$
```

Рис. 2.2: Компиляция и запуск программы lab9-1.asm

Затем я внес некоторые корректировки в код программы, включив дополнительную функцию `subcalcul` внутри `calcul` для расчёта формулы $f(g(x))$, при этом значение x по-прежнему вводится через клавиатуру, а функции $f(x) = 2x + 7$ и $g(x) = 3x - 1$ обрабатываются внутри этих функций.



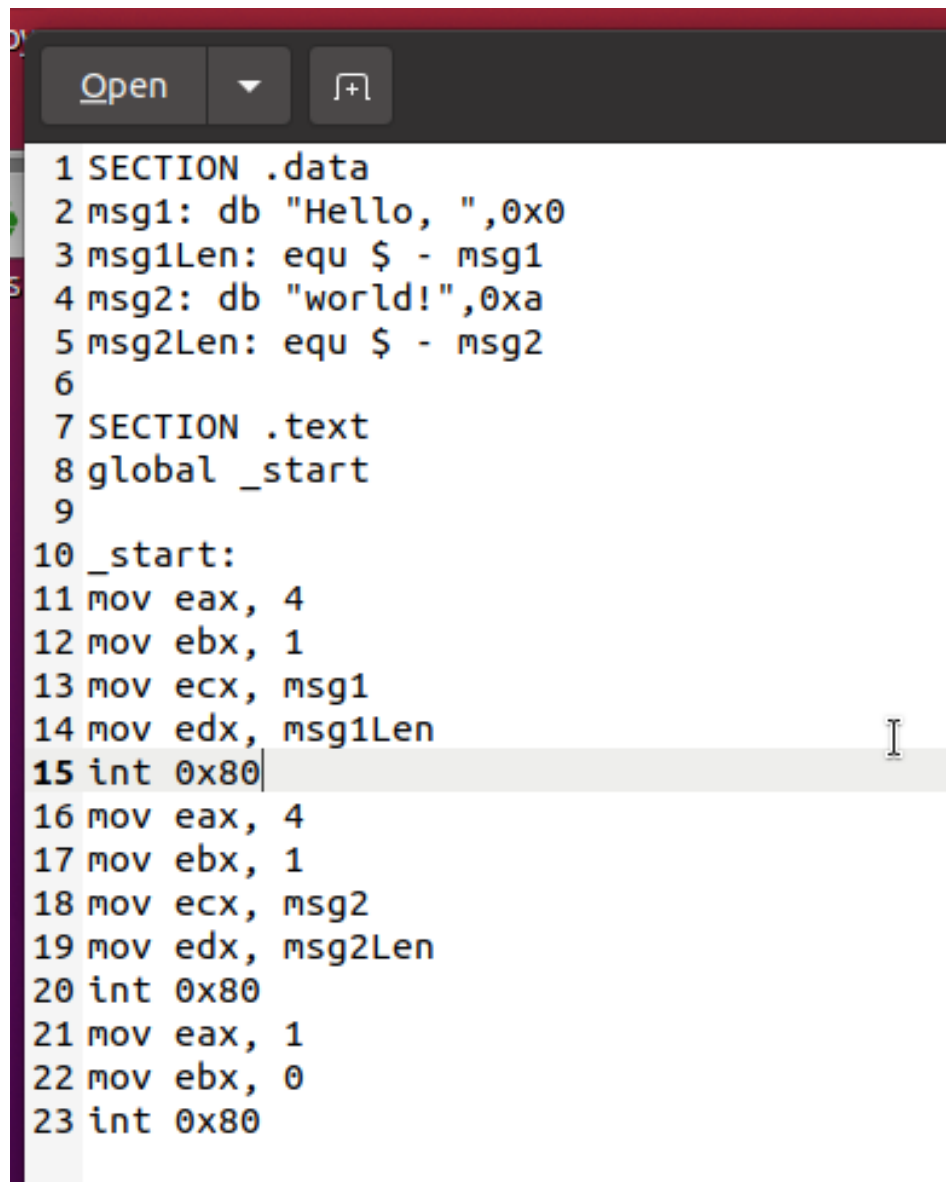
```
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2(3x-1)+7=',0
5
6 SECTION .bss
7 x: RESB 80
8 rez: RESB 80
9
10 SECTION .text
11 GLOBAL _start
12 _start:
13 mov eax, msg
14 call sprint
15 mov ecx, x
16 mov edx, 80
17 call sread
18 mov eax,x
19 call atoi
20 call _calcul ; Вызов подпрограммы _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: Код программы lab9-1.asm


```
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab09$  
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm  
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o  
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab09$ ./lab9-1  
Введите x: 3  
2(3x-1)+7=23  
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab09$
```

Рис. 2.4: Компиляция и запуск программы lab9-1.asm

Кроме того, я подготовил файл lab9-2.asm, содержащий код программы из Примера 9.2, который демонстрирует программу для вывода сообщения “Hello world!” на экран.



```
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: Код программы lab9-2.asm

Я добавил отладочную информацию с помощью ключа ‘-g’ для возможности работы с отладчиком GDB.

После этого я загрузил исполняемый файл в отладчик GDB и проверил функционирование программы, активировав её командой ‘run’ (или ‘r’).

```

huleroyun@Huler-Ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab09$
huleroyun@Huler-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/huleroyun/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 6381) exited normally]
(gdb)

```

Рис. 2.6: Компиляция и запуск программы lab9-2.asm в отладчике

Для тщательного анализа программы я установил точку останова на метке 'start', с которой стартует исполнение любой программы на ассемблере, и запустил программу для наблюдения. После этого я осмотрел дизассемблированный код программы, чтобы понять её структуру и работу.

```
huleroyn@Huler-Ubuntu: ~/work/arch-pc/lab09
<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/huleroyn/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 6381) exited normally]
(gdb) break _start
Breakpoint 1 at 0x08049000
(gdb) run
Starting program: /home/huleroyn/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

```
huleroyun@Huler-Ubuntu: ~/work/arch-pc/la
(gdb)
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: Дизассемблированный код в режиме интел

Чтобы проверить наличие брейкпоинта с меткой '_start', я применил команду 'info breakpoints' (или 'i b'). После этого я задал еще один брейкпоинт на адресе предпоследней команды 'mov ebx, 0x0'.

```
huleroyun@Huler-Ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>

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 6385 In: _start L?? PC: 0x8049000
(gdb)
(gdb)
(gdb) b *0x8049031 Breakpoint 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

Используя отладчик GDB, я мог наблюдать и редактировать содержимое памяти и регистров. Я выполнил пять шагов командой 'stepi' (или 'si'), следя за изменениями в регистрах.

```
huleroyn@Huler-Ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>

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 6385 In: start L?? PC: 0x8049005
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
0x8049005 in _start ()
(gdb)
```

Рис. 2.10: Изменение регистров

```
huleroyn@Huler-Ubuntu: ~/work/arch-pc/lab09
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>

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 6385 In: _start L?? PC: 0x8049016
(gdb) si
0x08049005 in _start ()
(gdb)
(gdb) si0x0804900a in _start ()
(gdb)
(gdb) si0x0804900f in _start ()
(gdb)
(gdb) si0x08049014 in _start ()
(gdb)
(gdb) si
0x08049016 in _start ()
(gdb)
```

Рис. 2.11: Изменение регистров

Чтобы просмотреть значение переменной `msg1`, я воспользовался соответствующей командой для извлечения необходимой информации.


```
huleroyun@Huler-Ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>

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 6385 In: _start L?? PC: 0x8049016
(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>:    "Lorld!\n"
(gdb) 
```

Рис. 2.12: Изменение значения переменной

Я также использовал команду 'set' для модификации значений в регистрах или ячейках памяти, указывая при этом нужный регистр или адрес. Мне удалось изменить первый символ переменной msg1.

The screenshot shows a GDB debugger window titled "huleroyn@Huler-Ubuntu: ~/work/arch-pc/lab09". The window is divided into several sections. At the top, a "Register group: general" section lists the values of general-purpose registers: `eax` (0x8), `ecx` (0x804a000), `edx` (0x8), `ebx` (0x1), `esp` (0xffffd1d0), `ebp` (0x0), `esi` (0x0), `edi` (0x0), and `eip` (0x8049016). Below this, a disassembly window shows the assembly code for the current instruction set, with the instruction at address 0x8049016 highlighted: `>0x8049016 <_start+22> mov eax,0x4`. The bottom section shows the GDB prompt with the command `(gdb) p/s ecx3` and its output `134520832`. Other GDB commands and outputs are visible, including `p/x ecx4` (0x804a000), `p/s edx5` (8), `p/t edx6` (1000), and `p/x $edx` (0x8).

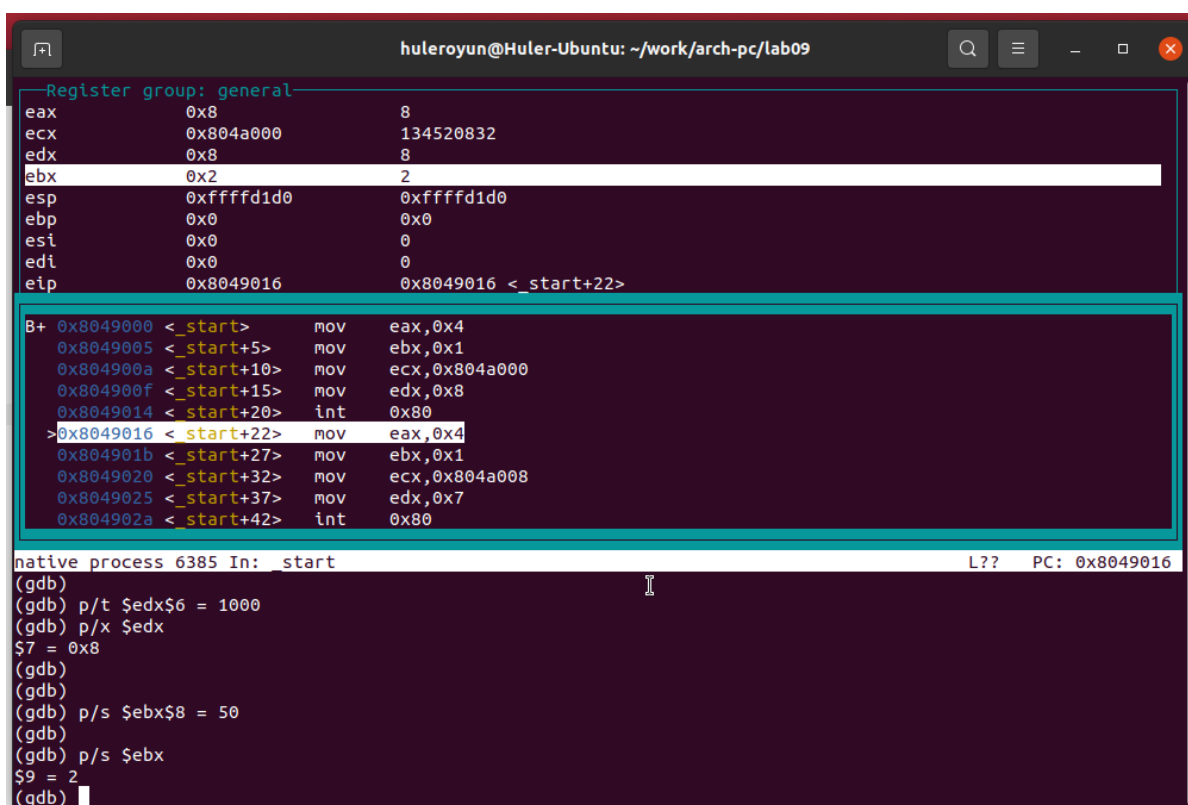
```
huleroyn@Huler-Ubuntu: ~/work/arch-pc/lab09
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>

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 6385 In: _start L?? PC: 0x8049016
(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

С помощью команды 'set' я изменил значение регистра `ebx` на требуемое.



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

Register	Value	Comment
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x2	2
esp	0xffffd1d0	0xffffd1d0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 <_start+22>

. The bottom section displays assembly code starting from address 0x8049000. The current instruction pointer (PC) is 0x8049016, which corresponds to the instruction "mov eax, 0x4" at offset +22 from the start of the function. Below the assembly code, the GDB prompt shows the execution of several commands: "(gdb) p/t \$edx\$6 = 1000", "(gdb) p/x \$edx", "\$7 = 0x8", "(gdb) p/s \$ebx\$8 = 50", "(gdb) p/s \$ebx", "\$9 = 2", and "(gdb)".

Рис. 2.14: Вывод значения регистра

Я скопировал файл lab8-2.asm, созданный в рамках лабораторной работы №8, который содержит код программы для вывода аргументов командной строки, и сформировал из него исполняемый файл.

Для запуска программы с аргументами в GDB я использовал опцию `-args`, загрузив исполняемый файл с заданными аргументами в отладчик.

Я установил брейкпоинт перед выполнением первой команды программы и начал ее выполнение.

Адрес вершины стека, содержащий количество аргументов командной строки (включая название программы), находится в регистре ESP. По этому адресу расположено число, показывающее количество аргументов. В моем случае было видно, что их пять, включая название программы lab9-3 и аргументы: аргумент1, аргумент2 и 'аргумент 3'.

Я также осмотрел другие записи стека. По адресу [ESP+4] расположен указа-

тель на имя программы в памяти. Адреса первого, второго и последующих аргументов находятся по адресам [ESP+8], [ESP+12] и так далее, с шагом в 4 байта, поскольку каждый следующий адрес отстоит на 4 байта от предыдущего ([ESP+4], [ESP+8], [ESP+12]).

```

huleroyn@Huler-Ubuntu: ~/work/arch-pc/lab09
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 0x080490e8
(gdb) run
Starting program: /home/huleroyn/work/arch-pc/lab09/lab9-3 argument 1 argument 2 argument\ 3

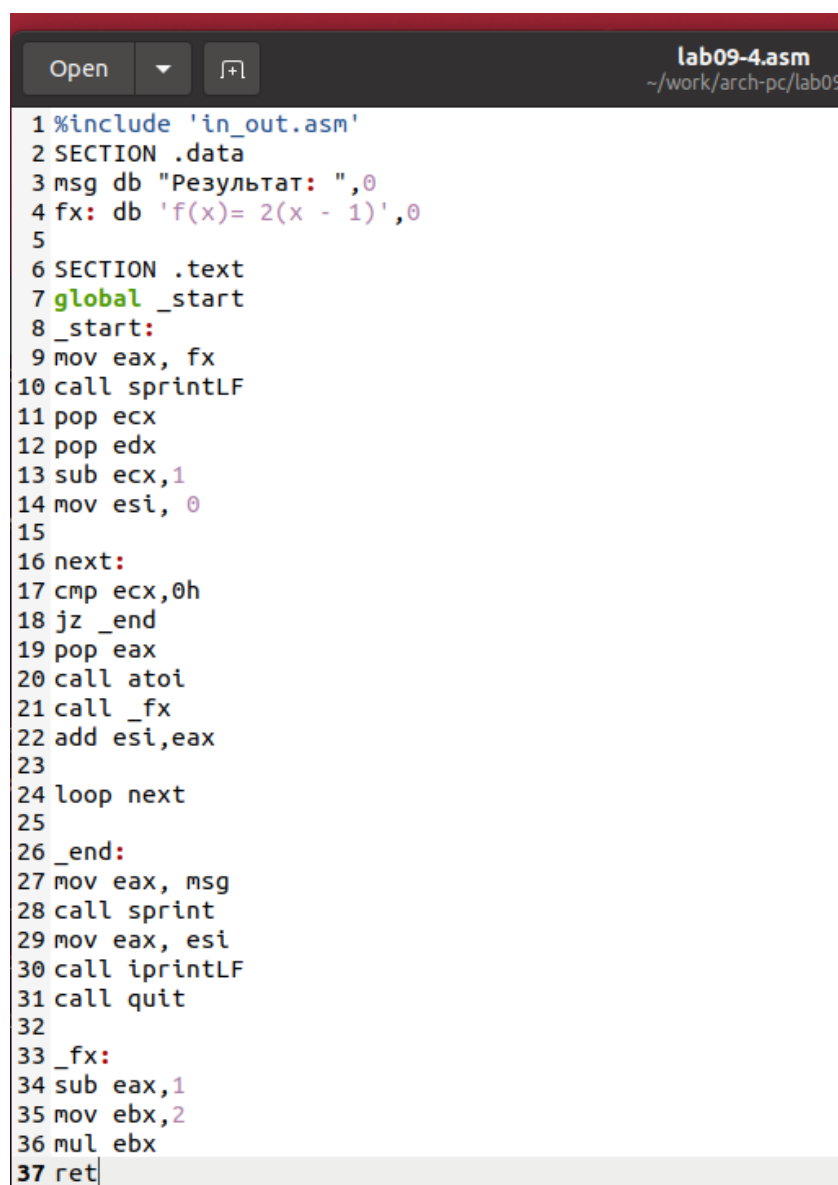
Breakpoint 1, 0x080490e8 in _start ()
(gdb) x/x $esp
0xffffd190: 0x00000006
(gdb)
0xffffd194: 0xffffd35a
(gdb) x/s *(void**)($esp + 4)
0xffffd35a: "/home/huleroyn/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd384: "argument"
(gdb) x/s *(void**)($esp + 12)
0xffffd38d: "1"
(gdb) x/s *(void**)($esp + 16)
0xffffd38f: "argument"
(gdb) x/s *(void**)($esp + 20)
0xffffd398: "2"
(gdb) x/s *(void**)($esp + 24)
0xffffd39a: "argument 3"
(gdb)

```

Рис. 2.15: Вывод значения регистра

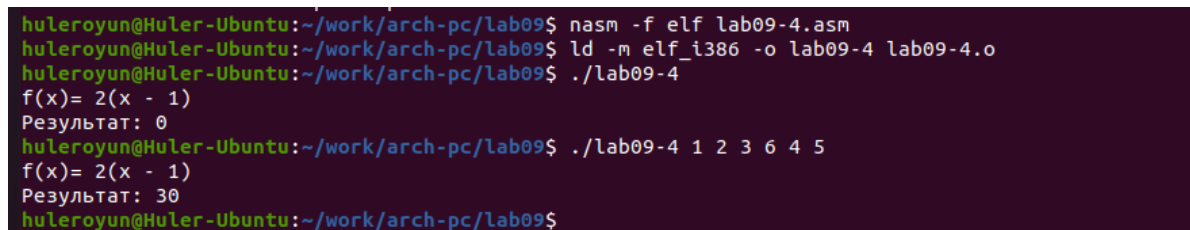
2.1 Задание для самостоятельной работы

Модифицировал код из восьмой лабораторной работы (Первое задание для индивидуального выполнения), создав подпрограмму для расчета значения функции $f(x)$.



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)= 2(x - 1)',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintf
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 sprintf
29 mov eax, esi
30 call iprintLF
31 call quit
32
33 _fx:
34 sub eax,1
35 mov ebx,2
36 mul ebx
37 ret
```

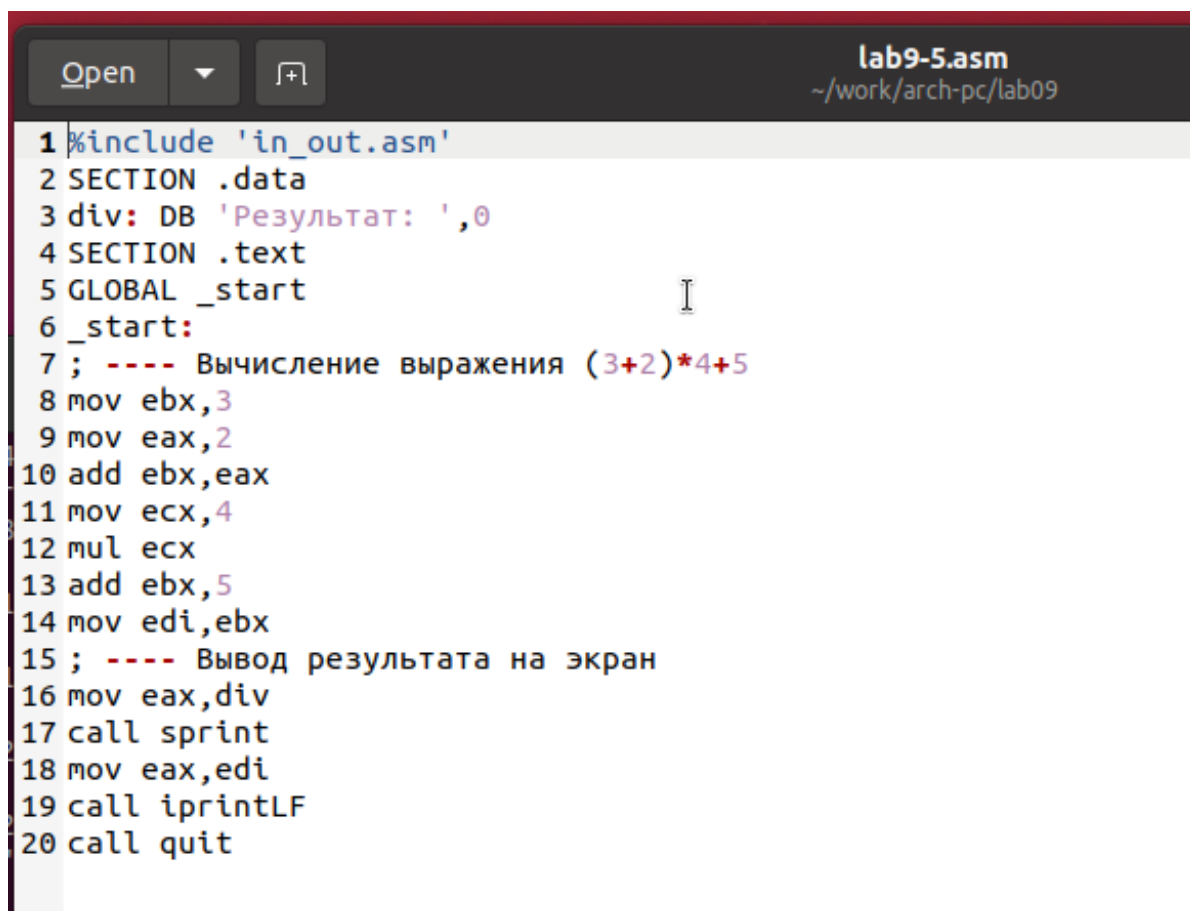
Рис. 2.16: Код программы lab9-4.asm



```
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab09$ ./lab09-4
f(x)= 2(x - 1)
Результат: 0
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab09$ ./lab09-4 1 2 3 6 4 5
f(x)= 2(x - 1)
Результат: 30
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab09$
```

Рис. 2.17: Компиляция и запуск программы lab9-4.asm

В представленном коде описан алгоритм для расчета формулы $(3 + 2) * 4 + 5$. Однако его исполнение приводит к некорректному итогу. Я выявил это, наблюдая за изменениями в регистрах при помощи отладчика GDB.



```
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 программы lab9-5.asm с ошибкой

```

huleroyn@Huler-Ubuntu: ~/work/arch-pc/lab09
eax      0x804a000      134520832
ecx      0x4           4
edx      0x0           0
ebx      0xa           10
esp      0xffffd1cc    0xffffd1cc
ebp      0x0           0x0
esi      0x0           0
edi      0xa           10
eip      0x804900f     0x804900f <sprint>

>0x804900f <sprint> push edx
B+ 0x80490e8 <_start> mov ebx,0x3
0x80490ed <_start+5> mov eax,0x2
0x80490f2 <_start+10> add ebx,eax
0x80490f4 <_start+12> mov ecx,0x400 <slen>
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

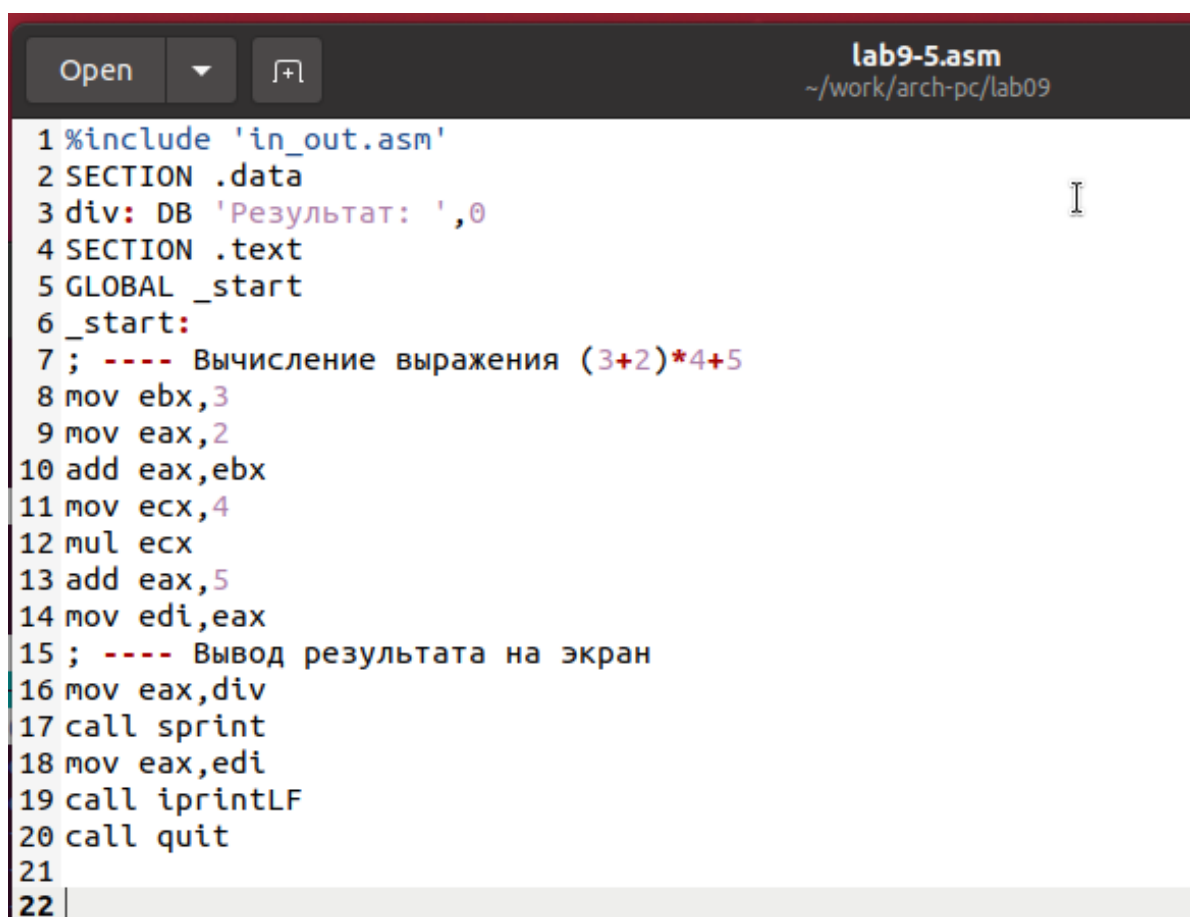
native process 6990 In: sprint
(gdb) si
(gdb) si
0x08049100 in _start ()
(gdb) si
0x08049105 in _start ()
(gdb) si
0x0804900f in sprint ()
(gdb) c
Continuing.
Результат: 10
[Inferior 1 (process 6990) exited normally]
(gdb)

```

Рис. 2.19: Отладка

Ошибка заключалась в неверном порядке аргументов команды add и в том, что в конце исполнения программы значение ebx переносится в edi вместо eax.

Исправленный код программы



```
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 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 программы lab9-5.asm исправлен


```
huleroyun@Huler-Ubuntu: ~/work/arch-pc/lab09

eax      0x19      25
ecx      0x4       4
edx      0x0       0
ebx      0x3       3
esp      0xffffd1d0 0xffffd1d0
ebp      0x0       0
esi      0x0       0
edi      0x19      25
eip      0x8049100 0x8049100 <_start+24>

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,edi

native process 7001 In: _start
(gdb) sNo process In:
(gdb) si
0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) si
0x08049100 in _start ()
(gdb) c
Continuing.
Результат: 25
[Inferior 1 (process 7001) exited normally]
(gdb) █
```

Рис. 2.21: Проверка работы

3 Выводы

Освоили работу с подпрограммами и отладчиком.