

Лабораторная работа №9

Понятие подпрограммы. Отладчик GDB

Налобин Михаил Дмитриевич

Содержание

1	Цель работы	6
2	Ход работы	7
3	Выводы	28

Список иллюстраций

2.1	Создание каталога lab09	7
2.2	Код программы lab9-1.asm	8
2.3	Запуск программы lab9-1	9
2.4	Измененная часть кода программы lab9-1.asm	9
2.5	Запуск измененной программы lab9-1	10
2.6	Создание lab9-2.asm	10
2.7	Код программы lab9-2.asm	10
2.8	Загрузка файла lab9-2 в отладчик	11
2.9	Проверка работы без брейкпоинта и с ним	11
2.10	Просмотр дисасимилированного кода программы	12
2.11	Вид дисасимилированного кода программы с Intel'овским синтаксисом	13
2.12	Режим псевдографики	13
2.13	Проверка и установка точек останова	14
2.14	Просмотр 6 строчки после _start	14
2.15	Просмотр 7 строчки после _start	15
2.16	Просмотр 8 строчки после _start	15
2.17	Просмотр 9 строчки после _start	16
2.18	Просмотр 10 строчки после _start	16
2.19	Просмотр содержимого регистров	17
2.20	Просмотр содержимого msg1 и msg2	17
2.21	Изменение первого символа msg1	17
2.22	Изменение второго символа msg2	18
2.23	Вывод значения регистра edx в разных форматах	18
2.24	Вывод символа 2 в регистре edx	19
2.25	Вывод числа 2 в регистре edx	19
2.26	Копирование lab8-2.asm	19
2.27	Создание исполняемого файла для lab9-3	19
2.28	Исследование стека с помощью отладки	20
2.29	Просмотр регистра esp	20
2.30	Просмотр позиций стека	21
2.31	Копирование sr.asm	21
2.32	Код преобразованной программы sr.asm	22
2.33	Выполнение программы sr.asm	23
2.34	Код программы lab9-4.asm	24
2.35	Выполнение команды lab9-4	24
2.36	Первая ошибка в lab9-4.asm	25

2.37	Первая ошибка в lab9-4.asm	26
2.38	Код измененной программы lab9-4.asm	27
2.39	Выполнение измененной команды lab9-4	27

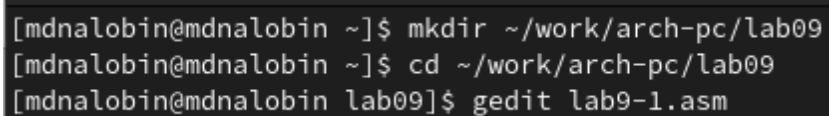
Список таблиц

1 Цель работы

Приобрести навык написания программ с использованием подпрограмм и знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Ход работы

Создали каталог lab09 и в данном каталоге файл lab9-1.asm (рис. 2.1).



```
[mdnlobin@mdnlobin ~]$ mkdir ~/work/arch-pc/lab09  
[mdnlobin@mdnlobin ~]$ cd ~/work/arch-pc/lab09  
[mdnlobin@mdnlobin lab09]$ gedit lab9-1.asm
```

Рис. 2.1: Создание каталога lab09

Переписали в него текст из Листинга 9.1. и, создав исполняемый файл, запустили lab9-1 (рис. 2.2 и рис. 2.3).

```

SECTION .data
    msg:    DB 'Введите x: ',0
    result: DB '2x+7=',0

SECTION .bss
    x:      RESB 80
    res:    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, result
    call sprint
    mov eax, [res]
    call iprintLF

    call quit

_calcul:
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [res], eax
    ret

```

Рис. 2.2: Код программы lab9-1.asm


```
[mdnalobin@mdnalobin lab09]$ nasm -f elf lab9-1.asm
[mdnalobin@mdnalobin lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[mdnalobin@mdnalobin lab09]$ ./lab9-1
Введите x: 5
2x+7=17
```

Рис. 2.3: Запуск программы lab9-1

После чего изменили текст файла lab9-1.asm, добавив еще одну подпрограмму, создали исполняемый файл и запустили его (рис. 2.4).

```
_calcul:
    call _subcalcul
    mov ebx,2
    mul ebx
    add eax,7
    mov [res],eax
    ret

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

Рис. 2.4: Измененная часть кода программы lab9-1.asm

```
[mdnalobin@mdnalobin lab09]$ nasm -f elf lab9-1.asm
[mdnalobin@mdnalobin lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[mdnalobin@mdnalobin lab09]$ ./lab9-1
Введите x: 2
2(3x-1)+7=17
```

Рис. 2.5: Запуск измененной программы lab9-1

Создали файл lab9-2.asm и заполнили его текстом из Листинга 9.2. (рис. 2.6 и рис. 2.7).

```
[mdnalobin@mdnalobin lab09]$ gedit lab9-2.asm
```

Рис. 2.6: Создание lab9-2.asm

```
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. 2.7: Код программы lab9-2.asm

Создаем исполняемый файл с применением ключа -g и загружаем его в отладчик gdb. Уже в самой оболочке проверяем работу программы и для более подробного анализа устанавливаем брейкпоинт на метку _start и запускаем снова (рис. 2.8 и рис. 2.9).

```
[mdnalobin@mdnalobin lab09]$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
[mdnalobin@mdnalobin lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[mdnalobin@mdnalobin lab09]$ gdb lab9-2
```

Рис. 2.8: Загрузка файла lab9-2 в отладчик

```
(gdb) run
Starting program: /home/mdnalobin/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 4295) exited normally]
(gdb) break _start
Breakpoint 1 at 0x4010e0: file lab9-2.asm, line 12.
(gdb) run
Starting program: /home/mdnalobin/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:12
12      mov eax,4
```

Рис. 2.9: Проверка работы без брейкпоинта и с ним

Далее смотрим дисассимилированный код программы с помощью команды disassemble, начиная с метки _start (рис. 2.10).

```

(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x004010e0 <+0>:      mov     $0x4,%eax
    0x004010e5 <+5>:      mov     $0x1,%ebx
    0x004010ea <+10>:     mov     $0x402118,%ecx
    0x004010ef <+15>:     mov     $0x8,%edx
    0x004010f4 <+20>:     int     $0x80
    0x004010f6 <+22>:     mov     $0x4,%eax
    0x004010fb <+27>:     mov     $0x1,%ebx
    0x00401100 <+32>:     mov     $0x402120,%ecx
    0x00401105 <+37>:     mov     $0x7,%edx
    0x0040110a <+42>:     int     $0x80
    0x0040110c <+44>:     mov     $0x1,%eax
    0x00401111 <+49>:     mov     $0x0,%ebx
    0x00401116 <+54>:     int     $0x80
End of assembler dump.

```

Рис. 2.10: Просмотр дисасемблированного кода программы

Затем переключили на отображение команд с синтаксисом intel, введя `set disassembly-flavor intel`, и снова применяем команду `disassemble`. Различие между синтаксисами АТТ и Intel заключается в конструкции: у АТТ первый после команды идет источник, а после приёмник; у Intel - Приёмник, источник (рис. 2.11).

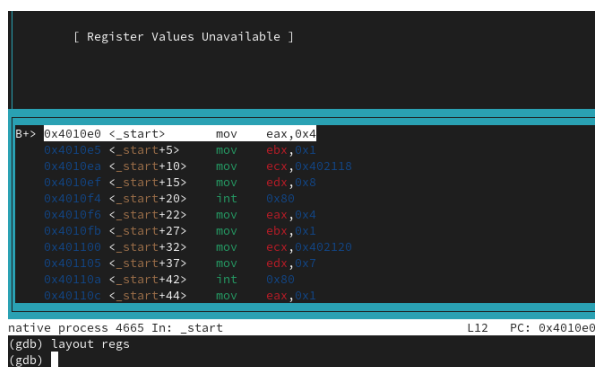
```

(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x004010e0 <+0>:      mov     eax,0x4
    0x004010e5 <+5>:      mov     ebx,0x1
    0x004010ea <+10>:     mov     ecx,0x402118
    0x004010ef <+15>:     mov     edx,0x8
    0x004010f4 <+20>:     int     0x80
    0x004010f6 <+22>:     mov     eax,0x4
    0x004010fb <+27>:     mov     ebx,0x1
    0x00401100 <+32>:     mov     ecx,0x402120
    0x00401105 <+37>:     mov     edx,0x7
    0x0040110a <+42>:     int     0x80
    0x0040110c <+44>:     mov     eax,0x1
    0x00401111 <+49>:     mov     ebx,0x0
    0x00401116 <+54>:     int     0x80
End of assembler dump.

```

Рис. 2.11: Вид дисасимилированного кода программы с Intel'овским синтаксисом

Включили режим псевдографики для более удобного анализа программы (рис. 2.12).



```

[ Register Values Unavailable ]

B> 0x4010e0 <_start>  mov     eax,0x4
    0x4010e5 <_start+5>  mov     ebx,0x1
    0x4010ea <_start+10> mov     ecx,0x402118
    0x4010ef <_start+15> mov     edx,0x8
    0x4010f4 <_start+20> int     0x80
    0x4010f6 <_start+22> mov     eax,0x4
    0x4010fb <_start+27> mov     ebx,0x1
    0x401100 <_start+32> mov     ecx,0x402120
    0x401105 <_start+37> mov     edx,0x7
    0x40110a <_start+42> int     0x80
    0x40110c <_start+44> mov     eax,0x1

native process 4665 In: _start      L12  PC: 0x4010e0
(gdb) layout regs
(gdb)

```

Рис. 2.12: Режим псевдографики

Проверили установленную точку останова командой `info breakpoints` (кратко `i` b), после установили еще одну точку останова и проверили это (рис. 2.13).

```

b+ 0x401111 <_start+49> mov ebx,0x0
native process 4665 In: _start L12 PC: 0x4010e0
(gdb) layout regs
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x004010e0 lab9-2.asm:12
breakpoint already hit 1 time
(gdb) b *0x401111
Breakpoint 2 at 0x401111: file lab9-2.asm, line 25.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x004010e0 lab9-2.asm:12
breakpoint already hit 1 time
2 breakpoint keep y 0x00401111 lab9-2.asm:25

```

Рис. 2.13: Проверка и установка точек останова

Выполнили 5 инструкций с использованием команды `stepi`, проследив за изменениями регистров, и заметили это у регистров `eax`, `ecx` и `edx` на рис. 14-2, 14-4 и 14-5 соответственно (рис. 2.14, рис. 2.15, рис. 2.16, рис. 2.17 и рис. 2.18).

```

--Register group: general--
eax      0x8      8
ecx      0x402118 4202776
edx      0x8      8
ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x4010f6 <_start+22>
eflags   0x202    [ IF ]

0x4010e5 <_start+5> mov ebx,0x1
0x4010ea <_start+10> mov ecx,0x402118
0x4010ef <_start+15> mov edx,0x8
0x4010f4 <_start+20> int 0x80
> 0x4010f6 <_start+22> mov eax,0x4
0x4010fb <_start+27> mov ebx,0x1
0x401100 <_start+32> mov ecx,0x402120
0x401105 <_start+37> mov edx,0x7
0x40110a <_start+42> int 0x80
0x40110c <_start+44> mov eax,0x1
b+ 0x401111 <_start+49> mov ebx,0x0

```

Рис. 2.14: Просмотр 6 строчки после `_start`

```

--Register group: general--
eax      0x4      4
ecx      0x402118 4202776
edx      0x8      8
ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x4010fb 0x4010fb <_start+27>
eflags   0x202    [ IF ]

0x4010e5 <_start+5>  mov     ebx,0x1
0x4010ea <_start+10> mov     ecx,0x402118
0x4010ef <_start+15> mov     edx,0x8
0x4010f4 <_start+20> int     0x80
0x4010f6 <_start+22> mov     eax,0x4
> 0x4010fb <_start+27> mov     ebx,0x1
0x401100 <_start+32> mov     ecx,0x402120
0x401105 <_start+37> mov     edx,0x7
0x40110a <_start+42> int     0x80
0x40110c <_start+44> mov     eax,0x1
b+ 0x401111 <_start+49> mov     ebx,0x0

```

Рис. 2.15: Просмотр 7 строчки после _start

```

--Register group: general--
eax      0x4      4
ecx      0x402118 4202776
edx      0x8      8
ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x401100 0x401100 <_start+32>
eflags   0x202    [ IF ]

0x4010e5 <_start+5>  mov     ebx,0x1
0x4010ea <_start+10> mov     ecx,0x402118
0x4010ef <_start+15> mov     edx,0x8
0x4010f4 <_start+20> int     0x80
0x4010f6 <_start+22> mov     eax,0x4
0x4010fb <_start+27> mov     ebx,0x1
> 0x401100 <_start+32> mov     ecx,0x402120
0x401105 <_start+37> mov     edx,0x7
0x40110a <_start+42> int     0x80
0x40110c <_start+44> mov     eax,0x1
b+ 0x401111 <_start+49> mov     ebx,0x0

```

Рис. 2.16: Просмотр 8 строчки после _start

```

--Register group: general--
eax      0x4      4
ecx      0x402120 4202784
edx      0x8      8
ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x401105 0x401105 <_start+37>
eflags   0x202    [ IF ]

0x4010e5 <_start+5>  mov     ebx,0x1
0x4010ea <_start+10> mov     ecx,0x402118
0x4010ef <_start+15> mov     edx,0x8
0x4010f4 <_start+20> int     0x80
0x4010f6 <_start+22> mov     eax,0x4
0x4010fb <_start+27> mov     ebx,0x1
0x401100 <_start+32> mov     ecx,0x402120
> 0x401105 <_start+37> mov     edx,0x7
0x40110a <_start+42> int     0x80
0x40110c <_start+44> mov     eax,0x1
b+ 0x401111 <_start+49> mov     ebx,0x0

```

Рис. 2.17: Просмотр 9 строчки после _start

```

--Register group: general--
eax      0x4      4
ecx      0x402120 4202784
edx      0x7      7
ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x40110a 0x40110a <_start+42>
eflags   0x202    [ IF ]

0x4010e5 <_start+5>  mov     ebx,0x1
0x4010ea <_start+10> mov     ecx,0x402118
0x4010ef <_start+15> mov     edx,0x8
0x4010f4 <_start+20> int     0x80
0x4010f6 <_start+22> mov     eax,0x4
0x4010fb <_start+27> mov     ebx,0x1
0x401100 <_start+32> mov     ecx,0x402120
0x401105 <_start+37> mov     edx,0x7
> 0x40110a <_start+42> int     0x80
0x40110c <_start+44> mov     eax,0x1
b+ 0x401111 <_start+49> mov     ebx,0x0

```

Рис. 2.18: Просмотр 10 строчки после _start

Посмотрели содержимое регистров с помощью команды `info registers` (кратко `ir`) (рис. 2.19).


```
(gdb) i r
eax            0x0            0
ecx            0x0            0
edx            0x0            0
ebx            0x0            0
esp            0xffffd0a0      0xffffd0a0
ebp            0x0            0x0
esi            0x0            0
edi            0x0            0
eip            0x4010e0        0x4010e0 <_start>
eflags         0x202          [ IF ]
cs             0x23           35
ss             0x2b           43
ds             0x2b           43
es             0x2b           43
fs             0x0            0
gs             0x0            0
```

Рис. 2.19: Просмотр содержимого регистров

Посмотрели значение переменной msg1 по имени и msg2 по адресу (рис. 2.20).

```
(gdb) x/1sb &msg1
0x402118 <msg1>:      "Hello, "
(gdb) x/1sb 0x402120
0x402120 <msg2>:      "world!\n\034"
```

Рис. 2.20: Просмотр содержимого msg1 и msg2

Изменили командой set первого символа в msg1 и для второго в msg2 (рис. 2.21 и рис. 2.22).

```
(gdb) set {char}0x402118='h'
(gdb) x/1sb 0x402118
0x402118 <msg1>:      "hello, "
```

Рис. 2.21: Изменение первого символа msg1

```
(gdb) set {char}0x402121='0'  
(gdb) x/1sb &msg2  
0x402120 <msg2>: "wOrld!\n\034"
```

Рис. 2.22: Изменение второго символа msg2

Вывели в различных форматах значение регистра edx (рис. 2.23).

```
(gdb) p $edx  
$6 = 7  
(gdb) p/x $edx  
$7 = 0x7  
(gdb) p/t $edx  
$8 = 111  
(gdb) p/a $edx  
$9 = 0x7
```

Рис. 2.23: Вывод значения регистра edx в разных форматах

С помощью команды set изменили значение ebx на символ 2, далее изменили значение на число 2. Так как в первом случае символ, а во втором число, то и выводится разное значение (рис. 2.24 и рис. 2.25).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$10 = 50
```

Рис. 2.24: Вывод символа 2 в регистре ebx

```
(gdb) set $ebx=2
(gdb) p/s $ebx
$11 = 2
```

Рис. 2.25: Вывод числа 2 в регистре ebx

Далее завершаем выполнение команды с помощью continue и выходим из GDB с помощью quit.

Копируем файл lab8-2.asm в наш каталог с именем lab9-3.asm и создаем исполняемый файл (рис. 2.26 и рис. 2.27).

```
[mdnlobin@mdnlobin lab09]$ cp ~/work/study/2023-2024/Архитектура\ компьютера/arch-
pc/labs/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab9-3.asm
```

Рис. 2.26: Копирование lab8-2.asm

```
[mdnlobin@mdnlobin lab09]$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
[mdnlobin@mdnlobin lab09]$ nasm -f elf lab9-3.asm
[mdnlobin@mdnlobin lab09]$ ld -m elf-i386 -o lab9-3 lab9-3.o
ld: error: unknown emulation: elf-i386
[mdnlobin@mdnlobin lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
```

Рис. 2.27: Создание исполняемого файла для lab9-3

С указанием аргументов загрузили исполняемый файл в отладчик, затем для исследования стека установили точку основы и запустили ее (рис. 2.28).

```
[mdnalobin@mdnalobin lab09]$ gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora Linux 13.2-3.fc38
Copyright (C) 2023 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 lab9-3...
(No debugging symbols found in lab9-3)
(gdb) b _start
Breakpoint 1 at 0x4011a8
(gdb) run
Starting program: /home/mdnalobin/work/arch-pc/lab09/lab9-3 аргумент1 аргумент 2 аргумент\ 3
```

Рис. 2.28: Исследование стека с помощью отладки

Просматриваем регистр esp, ибо в нем хранится число аргументов командой строки, включая имя программы и просматриваем позиции стека, изменяя шаг изменения адреса на 4 по причине того, что элемент стека занимает 4 байта (рис. 2.29 и рис. 2.30)

```
(gdb) x/x $esp
0xfffffd170: 0x00000005
```

Рис. 2.29: Просмотр регистра esp

```

0xffffd327:    "/home/mdnlobin/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd351:    "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd363:    "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd374:    "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd376:    "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:    <error: Cannot access memory at address 0x0>

```

Рис. 2.30: Просмотр позиций стека

##Самостоятельная работа

Копируем sr.asm из каталога lab08 в lab09 и изменяем таким образом, чтобы вычисление значения функции $f(x)$ было подпрограммой (рис. 2.31 и рис. 2.32)

```

[mdnlobin@mdnlobin lab09]$ cp ~/work/study/2023-2024/Архитектура\ компьютера/arch-
pc/labs/lab08/sr.asm ~/work/arch-pc/lab09/sr.asm
[mdnlobin@mdnlobin lab09]$ gedit sr.asm

```

Рис. 2.31: Копирование sr.asm

```

next:
    cmp ecx,0
    jz _end

    pop eax
    call atoi

    call _function

    call iprintLF
    add esi, eax

    loop next

_end:
    mov eax,msg3
    call sprint
    mov eax,esi
    call iprintLF
    call quit

_function:
    mov ebx, 4
    mul ebx
    dec ebx
    sub eax,ebx
    ret

```

Рис. 2.32: Код преобразованной программы sr.asm

Затем создаем исполняем файл и проверяем на правильную работу (рис. 2.33).

```
[mdnalobin@mdnalobin lab09]$ gedit sr.asm
[mdnalobin@mdnalobin lab09]$ nasm -f elf sr.asm
[mdnalobin@mdnalobin lab09]$ ld -m elf_i386 -o sr sr.o
[mdnalobin@mdnalobin lab09]$ ./sr 1 2 3
Функция:  $f(x)=4x-3$ 
Значения функции:
1
5
9
Результат: 15
```

Рис. 2.33: Выполнение программы sr.asm

Создаем файл lab9-4.asm, записываем в него содержимое Листинга 9.3., создаем исполняемый файл и проверяем работу (рис. 2.34 и рис. 2.35)

```

#include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:
    mov ebx,3
    mov eax,2
    add ebx,eax
    mov ecx,4
    mul ecx
    add ebx,5
    mov edi,ebx

    mov eax,div
    call sprint
    mov eax,edi
    call iprintLF

    call quit

```

Рис. 2.34: Код программы lab9-4.asm

```

[mdnalobin@mdnalobin lab09]$ nasm -f elf lab09-4.asm
[mdnalobin@mdnalobin lab09]$ ld -m elf_i386 -o lab09-4 lab09-4.o
[mdnalobin@mdnalobin lab09]$ ./lab09-4
Результат: 10

```

Рис. 2.35: Выполнение команды lab9-4

С помощью отладчика GDB анализируем изменения значений регистров, чтобы

определить место ошибок. В итоге обнаруживаем ошибку в строке с *mul ecx*, потому что до этого мы прибавляем к *ebx*, но команда *mul* может умножать лишь на *eax*, поэтому должны из строчки *add ebx,ecx* получить *add eax,ebx*. Следующая ошибка кроется в строчке *add ebx,5*, ведь ранее проводили операции над *eax* и именно *eax* несет в себе значение уже выполненных действий (рис. 2.36 и рис. 2.37)

Register group: general		
eax	0x8	8
ecx	0x4	4
edx	0x0	0
ebx	0x5	5
esp	0xffffd1b0	0xffffd1b0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x4011db	0x4011db <_start+19>
eflags	0x202	[IF]

0x4011c7	<quit+12>	ret
B+ 0x4011c8	<_start>	mov ebx, 0x3
0x4011cd	<_start+5>	mov eax, 0x2
0x4011d2	<_start+10>	add ebx, eax
0x4011d4	<_start+12>	mov ecx, 0x4
0x4011d9	<_start+17>	mul ecx
> 0x4011db	<_start+19>	add ebx, 0x5
0x4011de	<_start+22>	mov edi, ebx
0x4011e0	<_start+24>	mov eax, 0x4021f8
0x4011e5	<_start+29>	call 0x4010ef <sprint>
0x4011ea	<_start+34>	mov eax, edi

native process 9985 In: _start L?? PC: 0x4011db

Рис. 2.36: Первая ошибка в lab9-4.asm

```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x4011de 0x4011de <_start+22>
eflags   0x206    [ PF IF ]

0x4011c7 <quit+12>    ret
B+ 0x4011c8 <_start>   mov     ebx,0x3
0x4011cd <_start+5>   mov     eax,0x2
0x4011d2 <_start+10>  add     ebx,eax
0x4011d4 <_start+12>  mov     ecx,0x4
0x4011d9 <_start+17>  mul     ecx
0x4011db <_start+19>  add     ebx,0x5
> 0x4011de <_start+22> mov     edi,ebx
0x4011e0 <_start+24>  mov     eax,0x4021f8
0x4011e5 <_start+29>  call    0x4010ef <sprint>
0x4011ea <_start+34>  mov     eax,edi

native process 9985 In: _start L?? PC: 0x4011de
```

Рис. 2.37: Первая ошибка в lab9-4.asm

В этом этапе исправляем все найденные ошибки, создаем исполняемый файл и проверяем работу(рис. 2.38 и рис. 2.39)

```

%include      'in_out.asm'

SECTION .data
div:         DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:

    mov ebx,3
    mov eax,2
    add eax,ebx
    mov ecx,4
    mul ecx
    add eax,5
    mov edi,eax

    mov eax,div
    call sprint
    mov eax,edi
    call iprintLF

    call quit

```

Рис. 2.38: Код измененной программы lab9-4.asm

```

[mdnalobin@mdnalobin lab09]$ gedit lab9-4.asm
[mdnalobin@mdnalobin lab09]$ nasm -f elf lab9-4.asm
[mdnalobin@mdnalobin lab09]$ ld -m elf_i386 -o lab9-4 lab9-4.o
[mdnalobin@mdnalobin lab09]$ ./lab9-4
Результат: 25

```

Рис. 2.39: Выполнение измененной команды lab9-4

3 Выводы

В ходе данной длительной лабораторной работы приобрели навык, с помощью которого сможем писать программы с использованием подпрограмм, и познакомились с методами отладки с использованием GDB и его основными возможностями.

...