

Отчет по лабораторной работе №9

Дисциплина: архитектура компьютера

Горбунова Яна Сергеевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Релаксация подпрограмм в NASM	8
4.1.1	Отладка программ с помощью GDB	10
4.1.2	Добавление точек останова	12
4.1.3	Работа с данными программы в GDB	13
4.1.4	Обработка аргументов командной строки в GDB	15
4.2	Задание для самостоятельной работы	16
5	Выводы	21
6	Список литературы	22

Список иллюстраций

4.1	Запуск программы в отладчике	10
4.2	Запуск отладчика с брейкпоинтом	11
4.3	Режим псевдографики	12
4.4	Добавление второй точки останова	12
4.5	Просмотр содержимого регистров	13
4.6	Просмотр содержимого переменных двумя способами	13
4.7	Изменение содержимого переменных двумя способами	14
4.8	Просмотр значения регистра разными представлениями	14
4.9	Примеры использования команды set	15
4.10	Подготовка новой программы	15
4.11	Проверка работы стека	16
4.12	Измененная программа предыдущей лабораторной работы	17
4.13	Поиск ошибки в программе через пошаговую отладку	19
4.14	Проверка корректировок в программе	19

Список таблиц

1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Самостоятельное выполнение заданий по материалам лабораторной работы

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа:

- обнаружение ошибки; • поиск её местонахождения; • определение причины ошибки; • исправление ошибки.

Можно выделить следующие типы ошибок:

- синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • семантические ошибки — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата; • ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

4 Выполнение лабораторной работы

4.1 Релазиация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы №9 .

Копирую в файл код из листинга, компилирую и запускаю его, данная программа выполняет вычисление функции.

Изменяю текст программы, добавив в нее подпрограмму, теперь она вычисляет значение функции для выражения $f(g(x))$.

Код программы:

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

```
SECTION .data
```

```
msg: DB 'Введите x: ', 0
```

```
result: DB '2(3x-1)+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:
```

```
push eax
```

```
call _subcalcul
```

```
mov ebx, 2
```

```
mul ebx
```

```
add eax, 7
```

```
mov [res], eax
```

```
pop eax
```

```
ret
```

```

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

```

4.1.1 Отладка программ с помощью GDB

В созданный файл копирую программу второго листинга, транслирую с созданием файла листинга и отладки, компону и запускаю в отладчике (рис. -fig. 4.1).

```

--Register group: general--
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf10 0xffffcf10  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
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

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 5886 (asm) In: _start      L11  PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 4.1: Запуск программы в отладчике

Запустив программу командой `gdb`, я убедилась в том, что она работает исправно .

Для более подробного анализа программы добавляю брейкпоинт на метку `_start` и снова запускаю отладку (рис. -fig. 4.2).

```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf10  0xffffcf10  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
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

0x80490f2  add  BYTE PTR [eax],al
0x80490f4  add  BYTE PTR [eax],al
0x80490f6  add  BYTE PTR [eax],al
0x80490f8  add  BYTE PTR [eax],al
0x80490fa  add  BYTE PTR [eax],al
0x80490fc  add  BYTE PTR [eax],al
0x80490fe  add  BYTE PTR [eax],al
0x8049700  add  BYTE PTR [eax],al
0x8049702  add  BYTE PTR [eax],al
0x8049704  add  BYTE PTR [eax],al

native process 5886 (asm) In: _start L11 PC: 0x8049000
breakpoint already hit 1 time
(gdb) layout asm
(gdb) layout regs
(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 24.
(gdb) i b
Num   Type      Disp Enb Address  What
1     breakpoint keep y  0x8049000 lab9-2.asm:11
      breakpoint already hit 1 time
2     breakpoint keep y  0x8049031 lab9-2.asm:24
(gdb)

```

Рис. 4.2: Запуск отладчика с брейкпоинтом

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel *амд топчик* .

Различия между синтаксисом АТТ и Intel заключаются в порядке операндов (АТТ - Операнд источника указан первым. Intel - Операнд назначения указан первым), их размере (АТТ - размер операндов указывается явно с помощью суффиксов, непосредственные операнды предваряются символом \$; Intel - Размер операндов неявно определяется контекстом, как ах, еах, непосредственные операнды пишутся напрямую), именах регистров(АТТ - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов).

Включаю режим псевдографики для более удобного анализа программы (рис. -fig. 4.3).

```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf10 0xffffcf10  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
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

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 5886 (asm) In: _start L11 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint       keep y 0x8049000 lab9-2.asm:11
      breakpoint already hit 1 time
(gdb)

```

Рис. 4.3: Режим псевдографики

4.1.2 Добавление точек останова

Проверяю в режиме псевдографики, что брейкпоинт сохранился.

Устанавливаю еще одну точку останова по адресу инструкции (рис. -fig. 4.4).

```

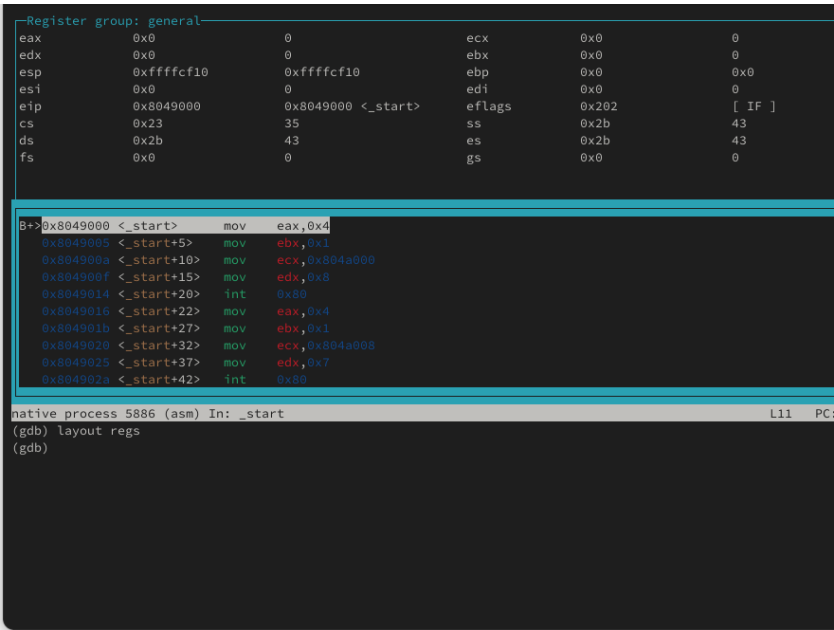
Breakpoint 1, _start () at lab9-2.asm:11
11      mov     eax, 4
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     $0x4,%eax
0x08049005 <+5>:      mov     $0x1,%ebx
0x0804900a <+10>:     mov     $0x804a000,%ecx
0x0804900f <+15>:     mov     $0x8,%edx
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) 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. 4.4: Добавление второй точки останова

4.1.3 Работа с данными программы в GDB

Просматриваю содержимое регистров командой `info registers` (рис. -fig. 4.5).



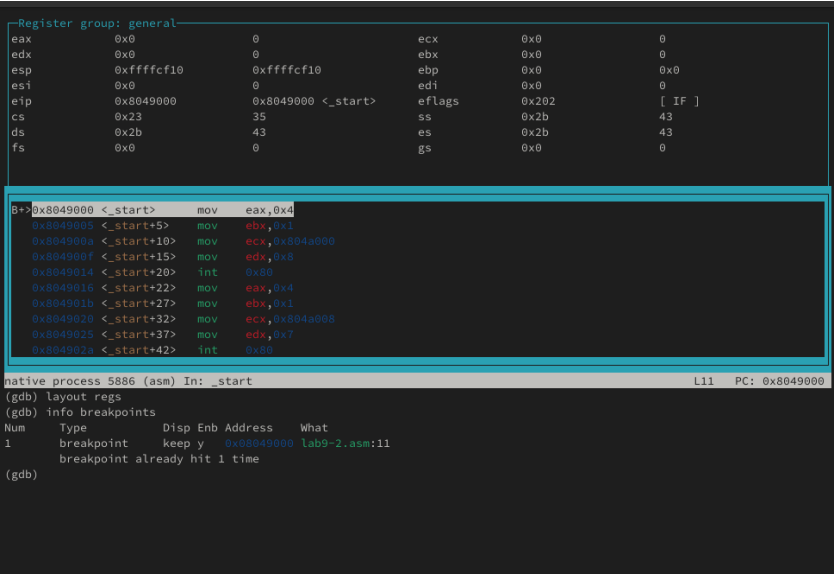
The screenshot shows the GDB interface. At the top, the 'Register group: general' is displayed with a table of register values. Below this, the assembly code for the current instruction is shown, along with the next few instructions. The assembly code is as follows:

Address	Disassembly
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 code, the GDB prompt shows the command `(gdb) layout regs` being executed.

Рис. 4.5: Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. -fig. 4.6).



The screenshot shows the GDB interface. At the top, the 'Register group: general' is displayed with a table of register values. Below this, the assembly code for the current instruction is shown, along with the next few instructions. The assembly code is as follows:

Address	Disassembly
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 code, the GDB prompt shows the command `(gdb) info breakpoints` being executed, displaying the following information:

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep y		0x8049000	lab9-2.asm:11

The GDB prompt also shows the command `(gdb) info breakpoints` being executed, displaying the following information:

breakpoint already hit 1 time

Рис. 4.6: Просмотр содержимого переменных двумя способами

Меняю содержимое переменных по имени и по адресу (рис. -fig. 4.7).

```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf10 0xffffcf10  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
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

0x80490f2  add  BYTE PTR [eax],al
0x80490f4  add  BYTE PTR [eax],al
0x80490f6  add  BYTE PTR [eax],al
0x80490f8  add  BYTE PTR [eax],al
0x80490fa  add  BYTE PTR [eax],al
0x80490fc  add  BYTE PTR [eax],al
0x80490fe  add  BYTE PTR [eax],al
0x8049100  add  BYTE PTR [eax],al
0x8049102  add  BYTE PTR [eax],al
0x8049104  add  BYTE PTR [eax],al

native process 5886 (asm) In: _start L11 PC: 0x8049000
breakpoint already hit 1 time
(gdb) layout asm
(gdb) layout regs
(gdb) layout regs
(gdb) break *0x08049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 24.
(gdb) i b
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:24
(gdb)

```

Рис. 4.7: Изменение содержимого переменных двумя способами

Вывожу в различных форматах значение регистра edx (рис. -fig. 4.8).

```

Register group: general
eax      0x8      8      ecx      0x804a000    134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffcf10 0xffffcf10  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

native process 5886 (asm) In: _start L17 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffcf10 0xffffcf10
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.8: Просмотр значения регистра разными представлениями

С помощью команды set меняю содержимое регистра ebx (рис. -fig. 4.9).

```

Register group: general
eax      0x8      8      ecx      0x804a000      134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffcf10 0xffffcf10  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,0x804a000
0x8049025 <_start+37>     mov     edx,0x7
0x804902a <_start+42>     int     0x80

native process 5886 (asm) In: _start      L17      PC: 0x8049016
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202      [ IF ]
cs       0x23      35
--Type <RET> for more, q to quit, c to continue without paging--
Quit
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "World!\n\034"
(gdb)

```

Рис. 4.9: Примеры использования команды set

4.1.4 Обработка аргументов командной строки в GDB

Копирую программу из предыдущей лабораторной работы в текущий каталог и создаю исполняемый файл с файлом листинга и отладки (рис. -fig. 4.10).

```

Register group: general
eax      0x8      8      ecx      0x804a000      134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffcf10 0xffffcf10  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,0x804a000
0x8049025 <_start+37>     mov     edx,0x7
0x804902a <_start+42>     int     0x80

native process 5886 (asm) In: _start      L17      PC: 0x8049016
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "World!\n\034"
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg2='x'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "xorld!\n\034"
(gdb)

```

Рис. 4.10: Подготовка новой программы

Запускаю программу с режиме отладки с указанием аргументов, указываю

брейкпоинт и запускаю отладку. Проверяю работу стека, изменяя аргумент команды просмотра регистра esp на +4, число обусловлено разрядностью системы, а указатель void занимает как раз 4 байта, ошибка при аргументе +24 означает, что аргументы на вход программы закончились. (рис. -fig. 4.11).

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd070 0xffffd070
ebp      0x0      0
esi      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

native process 10469 (asm) In: _start      L15    PC: 0x8049016
(gdb) p/t $ecx
$2 = 10000000010010100000000000000000
(gdb) p/s $edx
$3 = 8
(gdb) p/t $edx
$4 = 1000
(gdb) p/x $edx
$5 = 0x8
(gdb)

```

Рис. 4.11: Проверка работы стека

4.2 Задание для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. -fig. 4.12).


```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd070 0xffffd070
ebp      0x0      0x0
esi      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

native process 10469 (asm) In: _start L15 PC: 0x8049016
(gdb) set $ebx='2'
(gdb) p/s
$6 = 8
(gdb) p/s $ebx
$7 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$8 = 2
(gdb)

```

Рис. 4.12: Измененная программа предыдущей лабораторной работы

Код программы:

```

#include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 10x - 4", 0
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_start:
mov eax, msg_func
call sprintf

pop ecx
pop edx

```

```

sub ecx, 1
mov esi, 0

next:
cmp ecx, 0h
jz _end
pop eax
call atoi

call _calculate_fx

add esi, eax
loop next

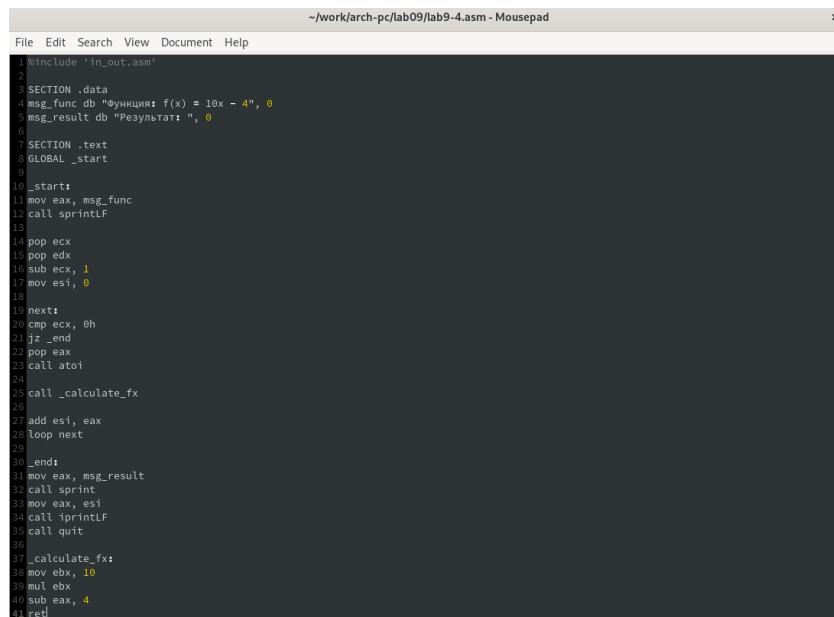
_end:
mov eax, msg_result
call sprint
mov eax, esi
call iprintLF
call quit

_calculate_fx:
mov ebx, 10
mul ebx
sub eax, 4

```

2. Запускаю программу в режиме отладчика и пошагово через si просматриваю изменение значений регистров через i r. При выполнении инструкции mul ecx можно заметить, что результат умножения записывается в регистр eax, но также меняет и edx. Значение регистра ebx не обновляется напря-

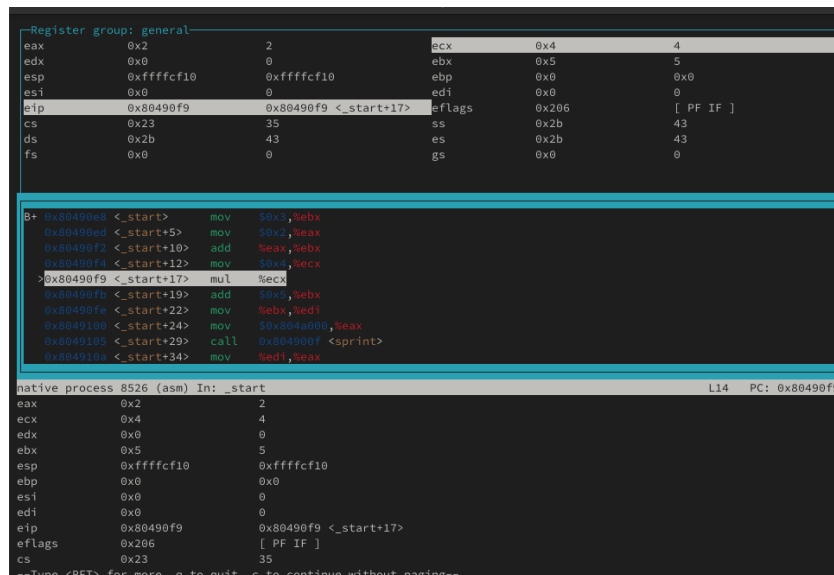
мую, поэтому результат программа неверно подсчитывает функцию (рис. -fig. 4.13).



```
1 %include "in_out.asm"
2
3 SECTION .data
4 msg_func db "Функция f(x) = 10x - 4", 0
5 msg_result db "Результат: ", 0
6
7 SECTION .text
8 GLOBAL _start
9
10 _start:
11 mov eax, msg_func
12 call sprintf
13
14 pop ecx
15 pop edx
16 sub ecx, 1
17 mov esi, 0
18
19 next:
20 cmp ecx, 0h
21 jz _end
22 pop eax
23 call atoi
24
25 call _calculate_fx
26
27 add esi, eax
28 loop next
29
30 _end:
31 mov eax, msg_result
32 call sprintf
33 mov eax, esi
34 call sprintf
35 call quit
36
37 _calculate_fx:
38 mov ebx, 10
39 mul ebx
40 sub eax, 4
41 ret
```

Рис. 4.13: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. -fig. 4.14).



```
Register group: general
eax 0x2 2 ecx 0x4 4
edx 0x0 0 ebx 0x5 5
esp 0xffffcf10 0xffffcf10 ebp 0x0 0x0
esi 0x0 0 edi 0x0 0
eip 0x80490f9 0x80490f9 <_start+17> eflags 0x206 [ PF 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
>0x80490f9 <_start+17> mul %ecx
0x804907b <_start+19> add $0x2,%ebx
0x80490fa <_start+22> mov %ebx,%edi
0x8049108 <_start+24> mov $0x804a800,%eax
0x8049105 <_start+29> call 0x804900f <sprintf>
0x804910a <_start+34> mov %edi,%eax

native process 8526 (asm) In: _start L14 PC: 0x80490f9
eax 0x2 2
ecx 0x4 4
edx 0x0 0
ebx 0x5 5
esp 0xffffcf10 0xffffcf10
ebp 0x0 0x0
esi 0x0 0
edi 0x0 0
eip 0x80490f9 0x80490f9 <_start+17>
eflags 0x206 [ PF IF ]
cs 0x23 35
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 4.14: Проверка корректировок в программе

Код измененной программы:

```
%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 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
```

5 Выводы

В результате выполнения данной лабораторной работы я приобрел навыки написания программ с использованием подпрограмм, а так же познакомился с методами отладки при помощи GDB и его основными возможностями.

6 Список литературы

1. Курс на ТУИС
2. Лабораторная работа №9
3. Программирование на языке ассемблера NASM Столяров А. В.