Отчет по лабораторной работе №9

По теме: Понятие подпрограммы. Отладчик GDB

Выполнил: Чубаев Кирилл Евгеньевич, НММбд-04-24

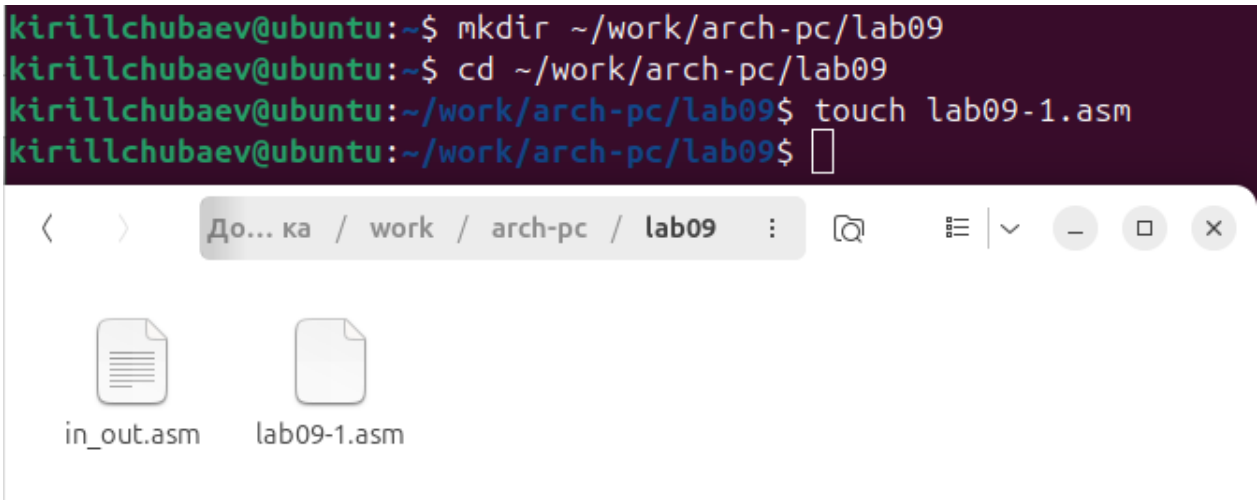
Содержание

Цель работы

Целью данной лабораторной работы является приобретение навыков написания программ с использованием подпрограмм, а также знакомство с методами отладки при помощи GDB и его основными возможностями

Ход выполнения лабораторной работы

1. Сначала я создал каталог lab09 и создал файл lab09-1.asm:



2. Я ввел код программы из листинга 9.1 в созданный файл и запустил программу:

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

```
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 6
2x+7=19
kirillchubaev@ubuntu:~/work/arch-pc/lab09$
```

3. Далее я изменил код программы так, чтобы она решала выражение f(g(x)). Программа работает корректно:

```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
prim1: DB 'f(x) = 2x+7',0
prim2: DB 'g(x) = 3x-1',0
result: DB 'f(g(x))= ',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax,prim1
call sprintLF
mov eax,prim2
call sprintLF
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:
call _subcalcul
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret
_subcalcul:
mov ebx,3
mul ebx
sub eax,1
ret
```

```
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ ./lab09-1
f(x) = 2x+7
g(x) = 3x-1
Введите x: 2
f(g(x))= 17
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ ./lab09-1
f(x) = 2x+7
g(x) = 3x-1
Введите x: 7
f(g(x))= 47
kirillchubaev@ubuntu:~/work/arch-pc/lab09$
```

4. Потом я создал файл lab09-2.asm и вписал туда код программы с помощью листинга 9.2:

```
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ touch lab09-2.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
```

5. Затем загрузил и запустил файл программы в отладчик gdb и поставил break-отметку на метку _start:

```
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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:
<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 lab09-2...
```

```
(gdb) run
Starting program: /home/kirillchubaev/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 37647) exited normally]
(gdb)
```

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/kirillchubaev/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9       mov eax, 4
(gdb)
```

6. Я внимательно изучил дисассимплированный код программы, начиная с метки:

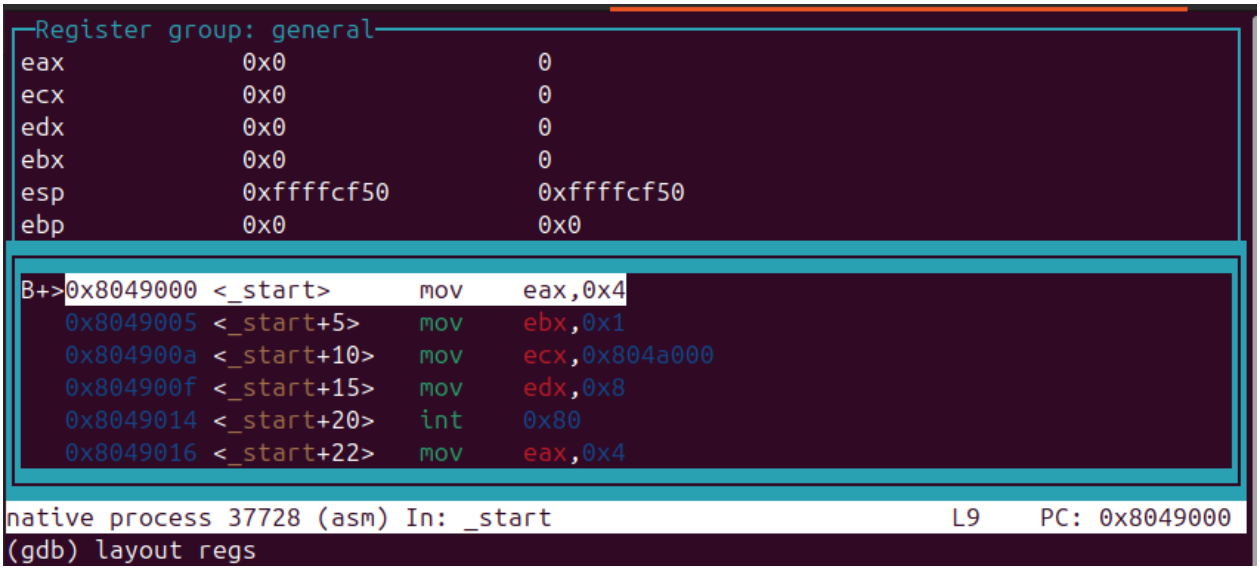
```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:     mov    $0x4,%eax
   0x08049005 <+5>:     mov    $0x1,%ebx
   0x0804900a <+10>:    mov    $0x804a000,%ecx
   0x0804900f <+15>:    mov    $0x8,%edx
   0x08049014 <+20>:    int    $0x80
   0x08049016 <+22>:    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.
```

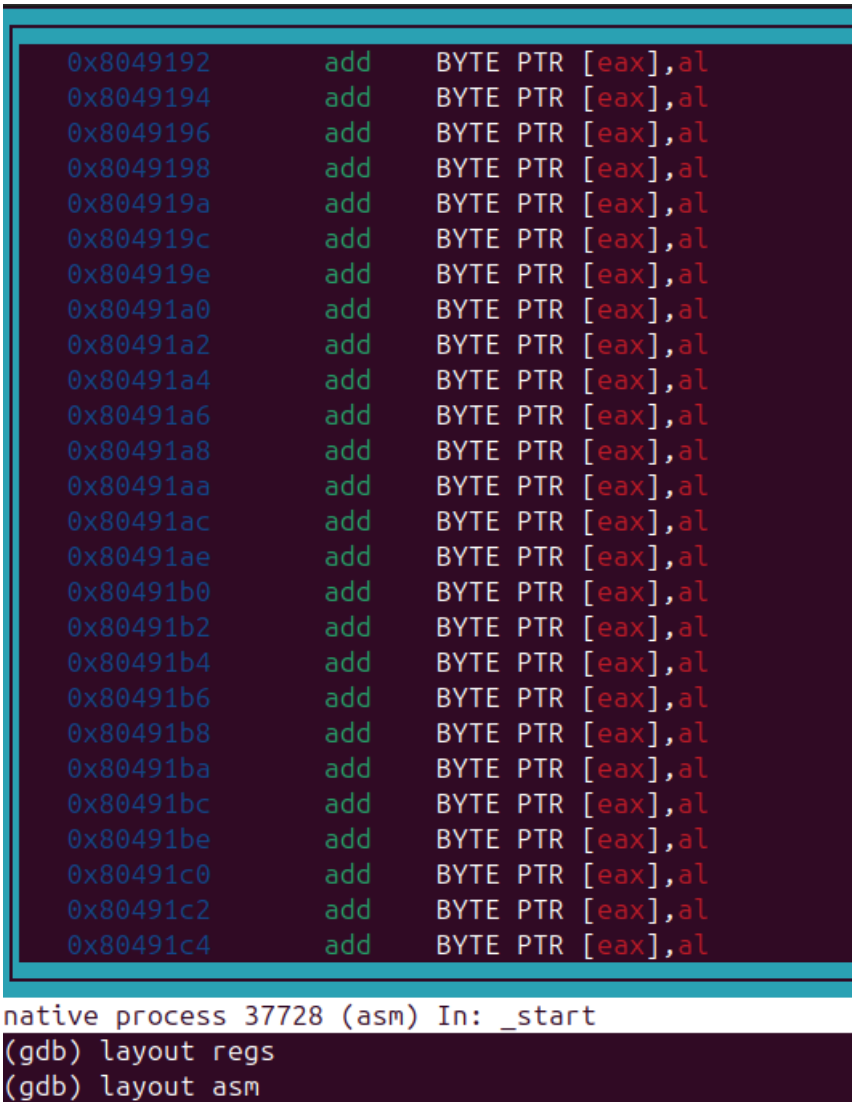
7. Далее с помощью специальных команд я переключился на intel’овское отображение синтаксиса:

```
(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.
```

Отличие заключается в том, что в диссамилированном отображении в командах используют символы “%” и “$”, а в Intel’овском отображении эти символы не используются.

8. Потом я включил режим псевдографики для удобного анализа программы:





9. Я посмотрел наличие меток с помощью специальной команды и установил еще одну метку по адресу инструкции:

```
(gdb) break *0x8049031
Note: breakpoint 2 also set at pc 0x8049031.
Breakpoint 3 at 0x8049031: file lab09-2.asm, line 20.
(gdb)
```

```
(gdb) i b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
2       breakpoint     keep y   0x08049031 lab09-2.asm:20
3       breakpoint     keep y   0x08049031 lab09-2.asm:20
(gdb)
```

10. С помощью команды info registers я посмотрел содержимое регистров:

```
eax            0x4                 4
ecx            0x804a000           134520832
edx            0x8                 8
ebx            0x1                 1
esp            0xffffcf50          0xffffcf50
ebp            0x0                 0x0
esi            0x0                 0
edi            0x0                 0
eip            0x8049014           0x8049014 <_start+20>
eflags         0x10202             [ IF RF ]
cs             0x23                35
--Type <RET> for more, q to quit, c to continue without paging--
```

11. Далее посмотрел значение переменной msg1 по специальному имени:

```
(gdb) x/1sb &msg1
0x804a000 <msg1>:       "Hello, "
```

12. Потом посмотрел значение второй переменной msg2 по адресу:

```
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:       "world!\n\034"
```

13. С помощью команды set я изменил значение переменной msg1:

```
(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:       "hhllo, "
(gdb)
```

14. По аналогичному принципу я изменил переменную msg2:

```
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb &msg2
0x804a008 <msg2>:       "Lor d!\n\034"
(gdb)
```

15. Затем я вывел значение регистра edx в двоичном, символьном и 16-ичном виде:

```
(gdb) p/f $msg1
$16 = void
(gdb) p/s $edx
$17 = 8
(gdb) p/t $edx
$18 = 1000
(gdb) p/c $edx
$19 = 8 '\b'
(gdb) p/x $edx
$20 = 0x8
(gdb)
```

16. Я изменил значение регистра ebx следующим способом:

```
(gdb) set $ebx='2
(gdb) p/s $ebx
$21 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$22 = 2
```

Команда выводит два разных значения, так как в первый раз вносится значение 2, а во втором случае регистр равен двум.

17. Я завершил работу с файлом в отладчике с помощью команд “c”, “si”, и “quit”:

```
(gdb) c
Continuing.
hhllo, Lor d!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb)
```

18. Далее я скопировал файл lab8-2.asm из лабораторной работы №8 и переименовал его. Запустил файл в отладчике, указал аргументы и запустил файл, поставив метку на _start:

```
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm
~/work/arch-pc/lab09/lab09-3.asm
```

```
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3
.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент
 2 'аргумент 3'
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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:
<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 lab09-3...
(gdb)
```

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/kirillchubaev/work/arch-pc/lab09/lab09-3 аргумент1 аргум
ент 2 аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.

Breakpoint 1, _start () at lab09-3.asm:5
5       pop ecx
```

19. Я проверил адрес вершины стека и убедился, что там хранится 5 элементов:

```
(gdb) x/x $esp
0xffffcf10:     0x00000005
```

20. Затем я посмотрел все позиции стека:

```
(gdb) x/s *(void**)($esp + 8)
0xffffd12f:     "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd141:     "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd152:     "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd154:     "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
```

По первому адресу хранится сам адрес, в остальных адресах хранятся элементы. Элементы расположены с интервалом в 4 единицы, так как стек может хранить только до 4 байт. Для того чтобы данные сохранялись нормально и без помех, компьютер использует новый стек для новой информации.

Выполнение самостоятельной работы

1. Сначала я создал файл для выполнения первого задания самостоятельной работы под названием lab09-test1.asm:

```
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ touch lab09-test1.asm
```

2. Затем преобразовал свой код программы из лабораторной работы №8 и реализовал вычисления как подпрограмму:

```
%include 'in_out.asm'

SECTION .data
prim DB 'f(x)=7*(x+1)',0
otv DB 'Результат: ',0

SECTION .text
GLOBAL _start

_start:
    pop ecx             ; Получаем количество аргументов (argc)
    pop edx             ; Пропускаем имя программы (argv[0])
    sub ecx, 1          ; Уменьшаем счетчик аргументов (не учитываем имя программы)

    mov esi, 0          ; Инициализируем сумму результатом 0

    mov eax, prim       ; Выводим строку с описанием функции
    call sprintLF

next:
    cmp ecx, 0          ; Если больше нет аргументов, завершаем цикл
    jz _end

    pop eax             ; Получаем следующий аргумент (в ASCII)
    call atoi           ; Преобразуем его в число
    call resh
    add esi, eax        ; Добавляем результат к общей сумме

    loop next
```

```
_end:
    mov eax, otv          ; Выводим строку с результатом
    call sprint

    mov eax, esi          ; Выводим итоговую сумму
    call iprintLF

    call quit

    resh:
    add eax, 1            ; Вычисляем x + 1
    mov ebx, 7            ; Умножаем на 7
    mul ebx
    ret
```

3. Создал исполняемый файл и запустил его, чтобы проверить правильность выполнения программы. Программа работает исправно:

```
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-test1.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-test1 lab09-t
est1.o
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ ./lab09-test1 3 5 7 9
f(x)=7*(x+1)
Результат: 196
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ ./lab09-test1 1 2 3 4
f(x)=7*(x+1)
Результат: 98
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ ./lab09-test1 1 2 3
f(x)=7*(x+1)
Результат: 63
kirillchubaev@ubuntu:~/work/arch-pc/lab09$
```

4. Для выполнения второго задания я создал файл lab09-test2.asm:

```
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ touch lab09-test2.asm
```

5. С помощью листинга 9.3 я написал код необходимой программы. Потом создал исполняемый файл и запустил его:

```
%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
```

```
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-test2.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-test2 lab09-test2.o

kirillchubaev@ubuntu:~/work/arch-pc/lab09$ ./lab09-test2
Результат: 10
kirillchubaev@ubuntu:~/work/arch-pc/lab09$
```

6. После выявления ошибки, которая связана с неправильным вычислением программы, я запустил её в отладчике:

```
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ gdb lab09-test2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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:
<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 lab09-test2...
(No debugging symbols found in lab09-test2)
(gdb) b _start
Breakpoint 1 at 0x80490e8
```

```
(gdb) run
Starting program: /home/kirillchubaev/work/arch-pc/lab09/lab09-test2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.

Breakpoint 1, 0x080490e8 in _start ()
(gdb) set disassembly-flavor intel
```

7. Затем я открыл регистры и внимательно проанализировал их. Я увидел, что некоторые регистры стоят не на своих местах и исправил это:

```
(gdb) disassemble _start

Dump of assembler code for function _start:
=> 0x080490e8 <+0>:     mov    ebx,0x3
   0x080490ed <+5>:     mov    eax,0x2
   0x080490f2 <+10>:    add    ebx,eax
   0x080490f4 <+12>:    mov    ecx,0x4
   0x080490f9 <+17>:    mul    ecx
   0x080490fb <+19>:    add    ebx,0x5
   0x080490fe <+22>:    mov    edi,ebx
   0x08049100 <+24>:    mov    eax,0x804a000
   0x08049105 <+29>:    call   0x804900f <sprint>
   0x0804910a <+34>:    mov    eax,edi
   0x0804910c <+36>:    call   0x8049086 <iprintLF>
   0x08049111 <+41>:    call   0x80490db <quit>
End of assembler dump.
```

8. После изменения регистров я запустил программу. Программа стала работать корректно и вывела в терминале ответ “25”:

```
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-test2.lst
lab09-test2.asm
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-test2 lab0
9-test2.o
kirillchubaev@ubuntu:~/work/arch-pc/lab09$ gdb lab09-test2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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:
<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 lab09-test2...
(gdb) run
Starting program: /home/kirillchubaev/work/arch-pc/lab09/lab09-test2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Результат: 25
[Inferior 1 (process 41655) exited normally]
(gdb)
```

Вывод

В ходе выполнения данной лабораторной работы я приобрел полезные навыки написания программ с использованием подпрограмм. Помимо этого, я познакомился с методами отладки при помощи GDB и с его основными возможностями.

Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.

2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.

3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander.org/.

4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.

5. *Newham C.* Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.

6. *Robbins A.* Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.

7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.

8. *Zarrelli G.* Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.

9. *Колдаев В. Д.*, *Лупин С. А.* Архитектура ЭВМ. — М. : Форум, 2018.

10. *Куляс О. Л.*, *Никитин К. А.* Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.

11. *Новожилов О. П.* Архитектура ЭВМ и систем. — М. : Юрайт, 2016.

12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.

13. *Робачевский А.*, *Немнюгин С.*, *Стесик О.* Операционная система UNIX. — 2-е изд. — БХВ-Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.

14. *Столяров А.* Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.

15. *Таненбаум Э.* Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).

16. *Таненбаум Э.*, *Бос Х.* Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).