Отчёт по лабораторной работе №9

Петлин Артём Дмитриевич

Содержание

Список иллюстраций

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. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции □(□) как подпрограмму.
2. В листинге 9.3 приведена программа вычисления выражения (3 + 2) □ 4 + 5. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

3 Теоретическое введение

3.1 Понятие об отладке

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 программазапускается, отрабатывает, 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.

3.2 Методы отладки

Наиболее часто применяют следующие методы отладки:

- создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);
- использование специальных программ-отладчиков.

Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам.
Пошаговое выполнение — это выполнение программы с остановкой после каждой строчки, чтобы программист мог проверить значения переменных и выполнить другие действия. Точки останова — это специально отмеченные места в программе, в которых программаотладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова:

- Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом);
- Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала

её значение, либо изменила его).

Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

3.3 Основные возможности отладчика GDB

GDB (GNU Debugger — отладчик проекта GNU) [1] работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки.
Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.
GDB может выполнять следующие действия:

- начать выполнение программы, задав всё, что может повлиять на её поведение;
- остановить программу при указанных условиях;
- исследовать, что случилось, когда программа остановилась;
- изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

3.4 Запуск отладчика GDB; выполнение программы; выход

Синтаксис команды для запуска отладчика имеет следующий вид:

```
gdb [опции] [имя_файла | ID процесса]
```

После запуска gdb выводит текстовое сообщение — так называемое «nice GDB logo». В следующей строке появляется приглашение (gdb) для ввода команд.
Далее приведён список некоторых команд GDB.
Команда run (сокращённо r) — запускает отлаживаемую программу в оболочке GDB.
Если точки останова не были установлены, то программа выполняется и выводятся сообщения:

```
(gdb) run
Starting program: test
Program exited normally.
(gdb)
```

Если точки останова были заданы, то отладчик останавливается на соответствующей команде и выдаёт номер точки останова, адрес и дополнительную информацию — текущую строку, имя процедуры, и др.
Команда kill (сокращённо k) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки:

```
Kill the program being debugged? (y or n) y
```

Если в ответ введено y (то есть «да»), отладка программы прекращается. Командой run её можно начать заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точки отлова (catchpoints) сохраняются.
Для выхода из отладчика используется команда quit (или сокращённо q):

```
(gdb) q
```

3.5 Дизассемблирование программы

Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом -g.
Посмотреть дизассемблированный код программы можно с помощью команды disassemble :

```
(gdb) disassemble _start
```

Существует два режима отображения синтаксиса машинных команд: режим Intel, используемый в том числе в NASM, и режим ATT (значительно отличающийся внешне). По умолчанию в дизассемблере GDB принят режим ATT. Переключиться на отображение команд с привычным Intel'овским синтаксисом можно, введя команду set disassembly-flavor intel.

3.6 Точки останова

Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»:

```
(gdb) break *<адрес>
(gdb) b <метка>
```

Информацию о всех установленных точках останова можно вывести командой info (кратко i):

```
(gdb) info breakpoints
(gdb) i b
```

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой disable: disable breakpoint
Обратно точка останова активируется командой enable:

```
enable breakpoint <номер точки останова>
```

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды delete:

```
(gdb) delete breakpoint <номер точки останова>
```

Ввод этой команды без аргумента удалит все точки останова. Информацию о командах этого раздела можно получить, введя

```
help breakpoints
```

3.7 Пошаговая отладка

Для продолжения остановленной программы используется команда continue (c) (gdb) c [аргумент]. Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число , которое указывает отладчику проигнорировать – 1 точку останова (выполнение остановится на -й точке).
Команда stepi (кратко sI) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию:

```
(gdb) si [аргумент]
```

При указании в качестве аргумента целого числа отладчик выполнит команду step раз при условии, что не будет точек останова или выполнение программы не прервётся по другим причинам.
Команда nexti (или ni) аналогична stepi, но вызов процедуры (функции) трактуется отладчиком как одна инструкция:

```
(gdb) ni [аргумент]
```

Информацию о командах этого раздела можно получить, введя

```
(gdb) help running
```

3.8 Работа с данными программы в GDB

Как уже упоминалось, отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных.
Посмотреть содержимое регистров можно с помощью команды info registers (или i r):

```
(gdb) info registers
```

Для отображения содержимого памяти можно использовать команду x/NFU , выдаёт содержимое ячейки памяти по указанному адресу.NFU задает формат, в котором выводятся данные.

Например,x/4uh 0x63450— это запрос на вывод четырёх полуслов (h) из памяти в формате беззнаковых десятичных целых (u), начиная с адреса 0x63450.
Чтобы посмотреть значения регистров используется команда print /F (сокращенно p). Перед именем регистра обязательно ставится префикс $. Например, команда p/x $ecx выводит значение регистра в шестнадцатеричном формате.
Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. При этом перед именем регистра ставится префикс $, а перед адресом нужно указать в фигурных скобках тип данных (размер сохраняемого значения; в качестве типа данных можно использовать типы языка Си).
Справку о любой команде gdb можно получить, введя

```
(gdb) help [имя_команды]
```

3.9 Понятие подпрограммы

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом.

Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

3.9.1 Инструкция call и инструкция ret

Для вызова подпрограммы из основной программы используется инструкция call, которая заносит адрес следующей инструкции в стек и загружает в регистр eip адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы.

Подпрограмма завершается инструкцией ret, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией call, и заносит его в eip. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией call. Подпрограмма может вызываться как из внешнего файла, так и быть частью основной программы.

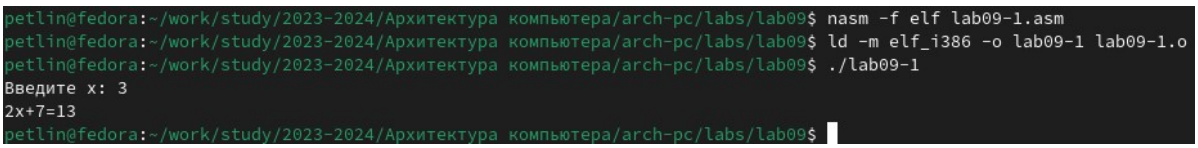
Важно помнить, что если в подпрограмме занести что-то в стек и не извлечь,то на вершине стека окажется не адрес возврата и это приведёт к ошибке выхода из подпрограммы. Кроме того, надо помнить, что подпрограмма без команды возврата не вернётся в точку вызова, а будет выполнять следующий за подпрограммой код, как будто он является её продолжением.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

```
petlin@fedora:~$ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab09
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ touch lab09-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ ls
lab09-1.asm  presentation  report
```

```
lab09-1.asm          [-M--]  0 L:[  1+35  36/ 36] *(708 / 708b) <EOF>
%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:
;---------------------------------------
; Основная программа
;---------------------------------------
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax,x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax,result
call sprint
mov eax,[res]
call iprintLF
call quit
;---------------------------------------
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы
```



Переходим в каталог для выполнения лабораторной работы № 9 и создаём файл lab09-1.asm, в который вписываем текст программы из листинга 9.1. Создаём исполняемый файл и проверяем его работу.

```
lab09-1.asm        [----] 17 L:[  1+38  39/ 41] *(748 / 797b) 0010 0x00A
%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:
;---------------------------------------------
; Основная программа
;---------------------------------------------
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax,x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax,result
call sprint
mov eax,[res]
call iprintLF
call quit
;---------------------------------------------
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
    call _subcalcul
    mov ebx,2
    mul ebx
    add eax,7
    mov [res],eax
    _subcalcul:
<------>mov ebx,3
<------>mul ebx
<------>sub eax,1
<------>ret ; выход из подпрограммы
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf lab09-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ ./lab09-1
Введите x: 4
2(3x-1)+7=29
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$
```

Изменяем текст программы, добавив подпрограмму в подпрограмму.

4.2 Отладка программам с помощью GDB

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ touch lab09-2.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ ls
in_out.asm  lab09-1  lab09-1.asm  lab09-1.o  lab09-2.asm  presentation  report
```

```
lab09-2.asm        [-M--]  0 L:[  1+21  22/ 22] *(311 / 311b) <EOF>
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
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 15.2-2.fc40
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-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 lab09-2...
(gdb) run
Starting program: /home/petlin/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
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 47.71 K separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 3269) exited normally]
```

Создаём файл lab09-2.asm, в который вписываем текст программы из листинга 9.2. Создаем исполняемый файл с использованием откладчика GDB. Проверяем работу программы, запустив ее в оболочке GDB с помощью команды run.

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/petlin/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09/lab09-2

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

Для более подробного анализа программы устанавливаем брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запускаем её.

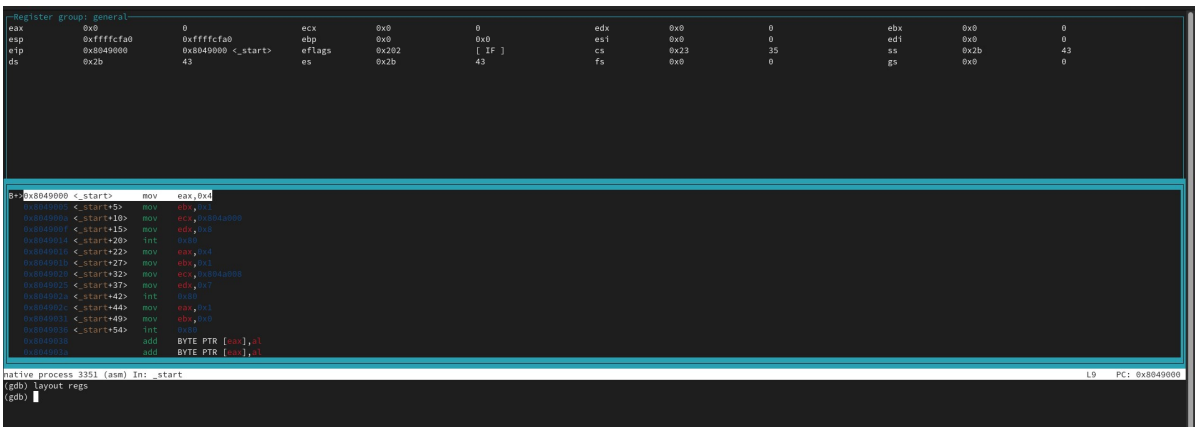
```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:     mov    $0x4,%eax
   0x08049005 <+5>:     mov    $0x1,%ebx
   0x0804900a <+10>:    mov    $0x804a000,%ecx
   0x0804900f <+15>:    mov    $0x8,%edx
   0x08049014 <+20>:    int    $0x80
   0x08049016 <+22>: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

Смотрим дисассимилированный код программы с помощью команды disassemble начиная с метки _start.

Переключаемся на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel.

Различия:

Порядок перечисления операндов и обозначение регистров в ATT используются символ "%



Включаем режим псевдографики для более удобного анализа программы.

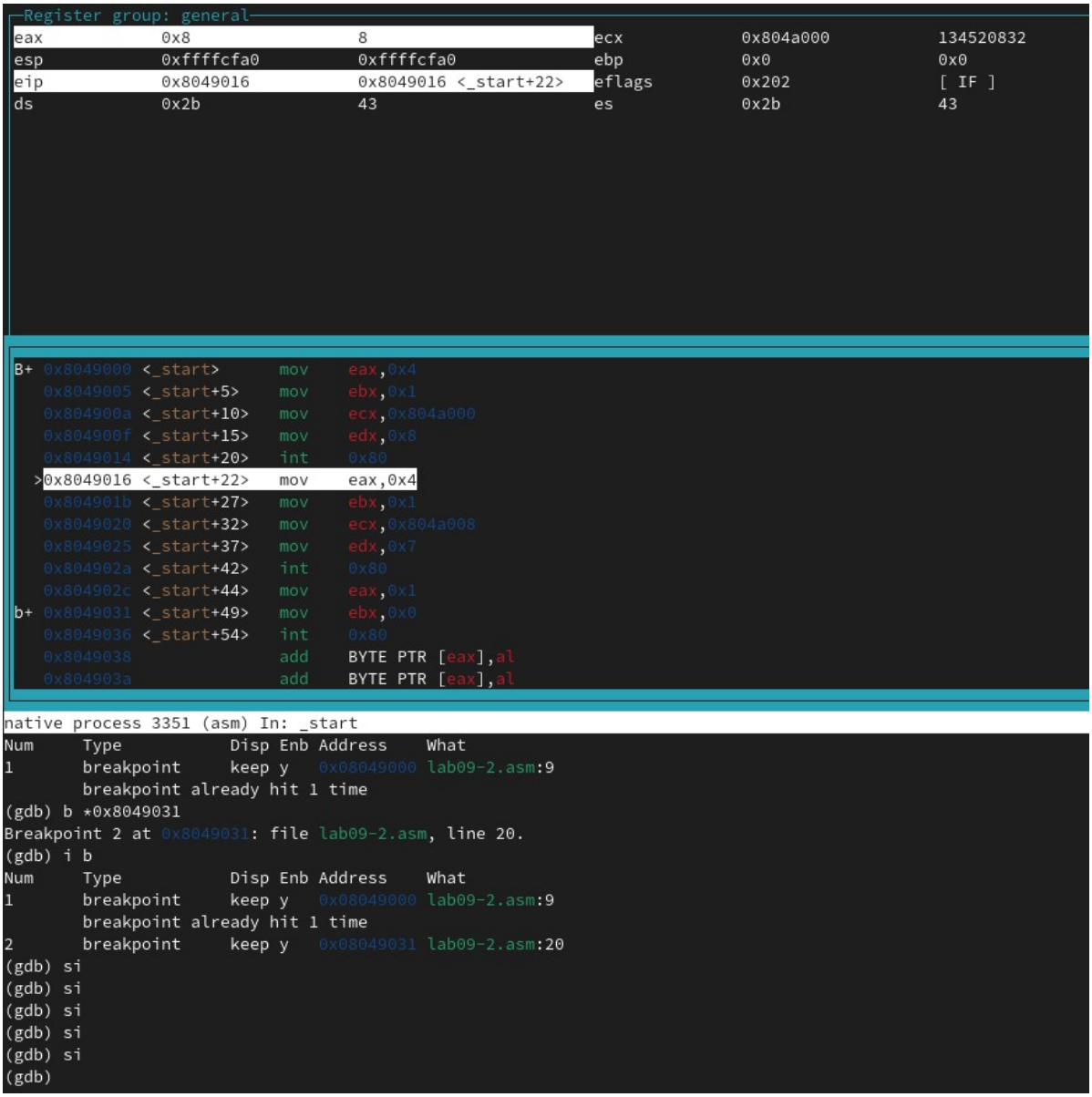
```
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
```

На предыдущих шагах мы установили точка останова по имени метки (_start).

Проверяем это с помощью команды info breakpoints.

```
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(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
(gdb)
```

Устанавливаем еще одну точку останова по адресу инструкции.



Выполните 5 инструкций с помощью команды stepi. Изменяются регистры ebx, ecx, edx, eax, eip.

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

Смотрим значение переменной msg1 по имени

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

Смотрим значение переменной msg2 по адресу

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

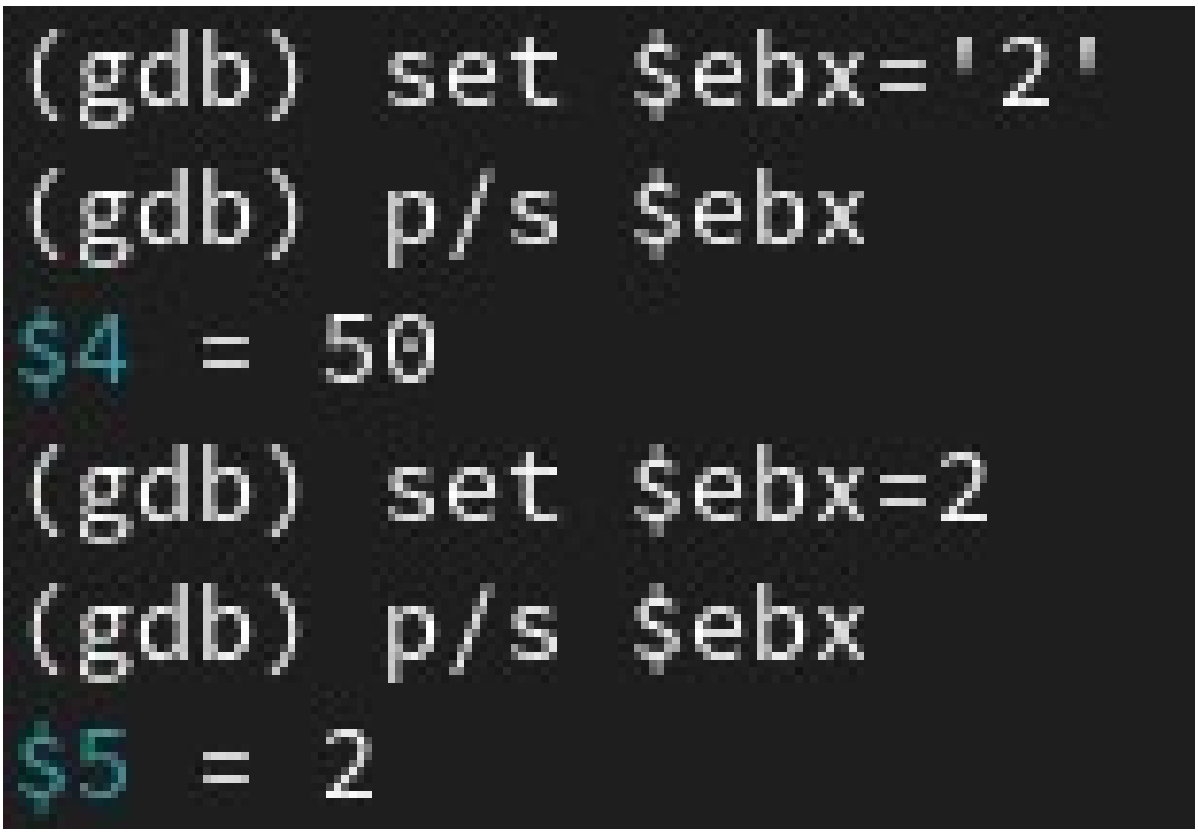
Изменяем первый символ переменной msg1

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

Изменяем первый символ переменной msg2

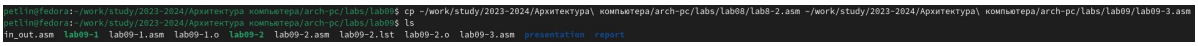
```
(gdb) p/t $edx
$1 = 1000
(gdb) p/s $edx
$2 = 8
(gdb) p/x $edx
$3 = 0x8
```

Смотрим значения регистра edx в различных форматах.



С помощью команды set изменяем значение разными способами регистра ebx. Во второй раз команда без кавычек присваивает регистру вводимое значение, поэтому вывод различен.

Завершаем выполнение программы и выходим из GDB.



Копируем файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm.

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ gdb --args lab09-3 2 5 7
GNU gdb (Fedora Linux) 15.2-2.fc40
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-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 lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/petlin/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09/lab09-3 2 5 7

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

Breakpoint 1, _start () at lab09-3.asm:5
5       pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffcf90:     0x00000004
(gdb)
```

Создаём исполняемый файл, указываем аргументы и запускаем его в оболочке GDB. Устанавливаем точку останова перед первой инструкцией в программе и запускаем ее. Смотрим количество аргументов.

```
(gdb) x/s *(void**)($esp + 4)
0xffffd151:     "/home/petlin/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd1ba:     "2"
(gdb) x/s *(void**)($esp + 12)
0xffffd1bc:     "5"
(gdb) x/s *(void**)($esp + 16)
0xffffd1be:     "7"
(gdb) x/s *(void**)($esp + 20)
0x0:    <error: Cannot access memory at address 0x0>
```

Смотрим позиции стека. Шаг изменения адреса равен 4, потому что регистры имеют размерность 4 байта.

4.3 Задание для самостоятельной работы

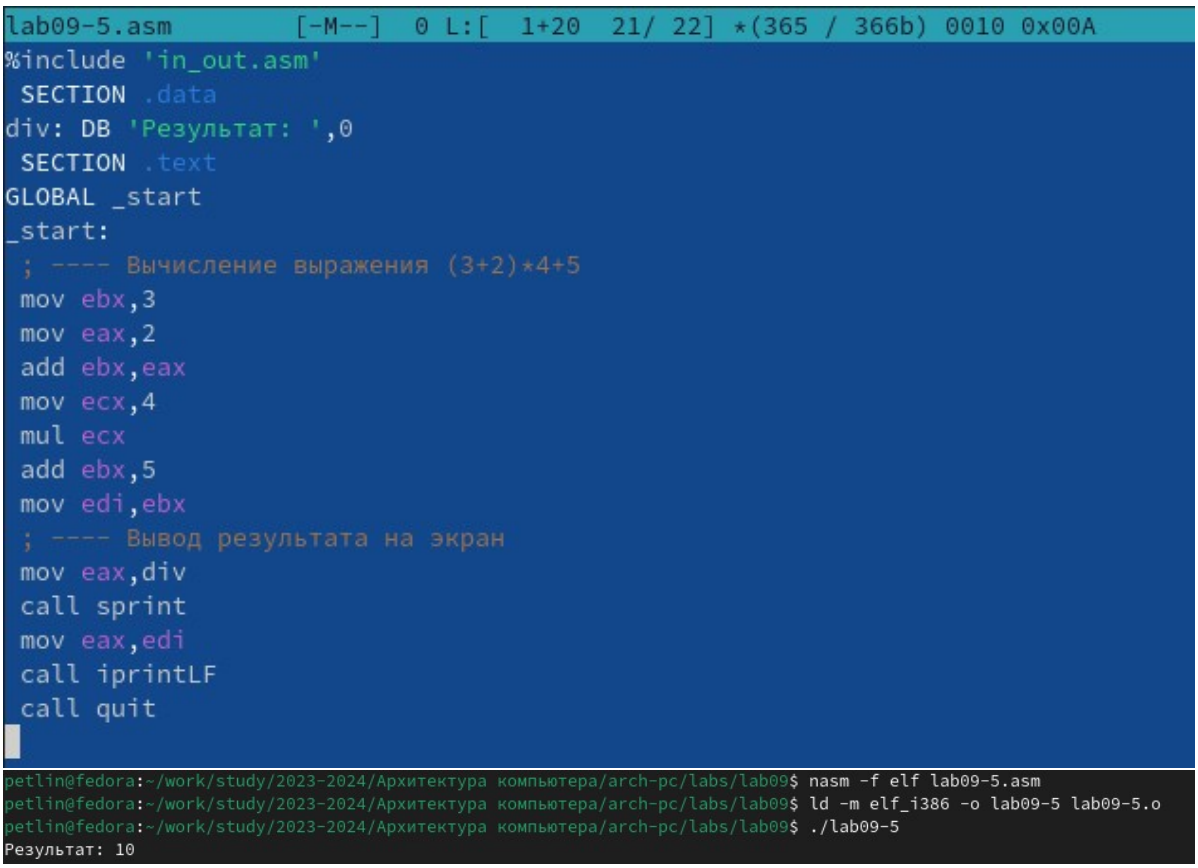
```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ cp ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab08/lab8-4.asm ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab09/lab09-4.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ls
in_out.asm  lab8-1  lab8-1.asm  lab8-1.o  lab8-2  lab8-2.asm  lab8-2.o  lab8-3  lab8-3.asm  lab8-3.o  lab8-4  lab8-4.asm  lab8-4.o  presentation  report
```

```
lab09-4.asm        [-M--]  0 L:[  1+29  30/ 30] *(411 / 411b) <EOF>
%include 'in_out.asm'
SECTION .data
 msg1 db "Введите x: ",0
 msg2 db "4x-3 = ",0
SECTION .bss
 x: RESB 80
 tmp: RESB 80
SECTION .text
 global _start
_start:
 mov eax,msg1
 call sprint
 mov ecx,x
 mov edx,80
 call sread
 mov eax,x
 call atoi
 call _calcul
 mov eax,msg2
 call sprint
 mov eax,[tmp]
 call iprintLF
 call quit
    _calcul:
    mov ebx,4
    mul ebx
    sub eax,3
    mov [tmp],eax
    ret
```

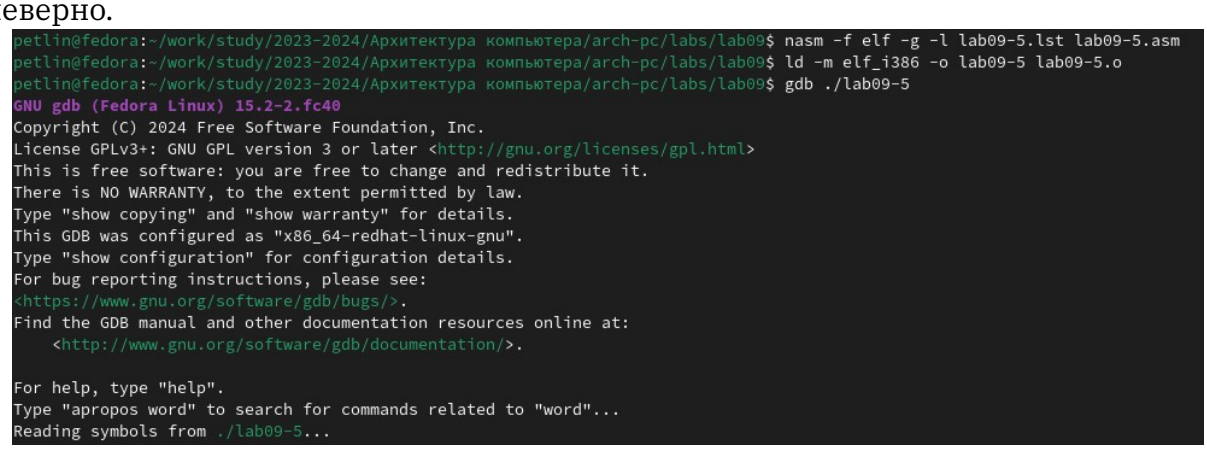
```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf lab09-4.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ ./lab09-4
Введите x: 5
4x-3 = 17
```

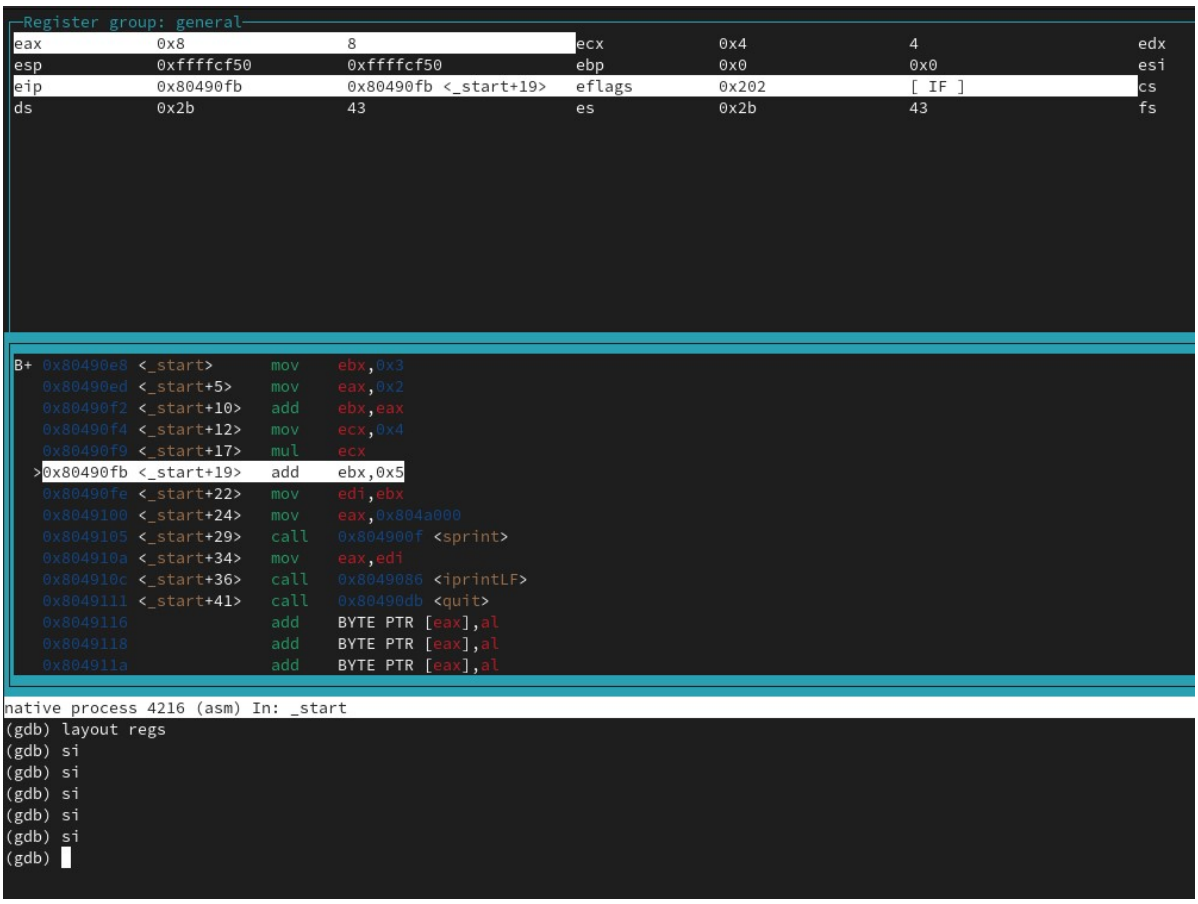
Копируем файл с текстом программы для задания для самостоятельной работы из лабораторной работы №8. Изменяем текст программы, реализовав вычисление функции как подпрограмму. Создаем исполняем файл и проверяем его работу. Программа работает корректно.

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09$ touch lab09-5.asm
```

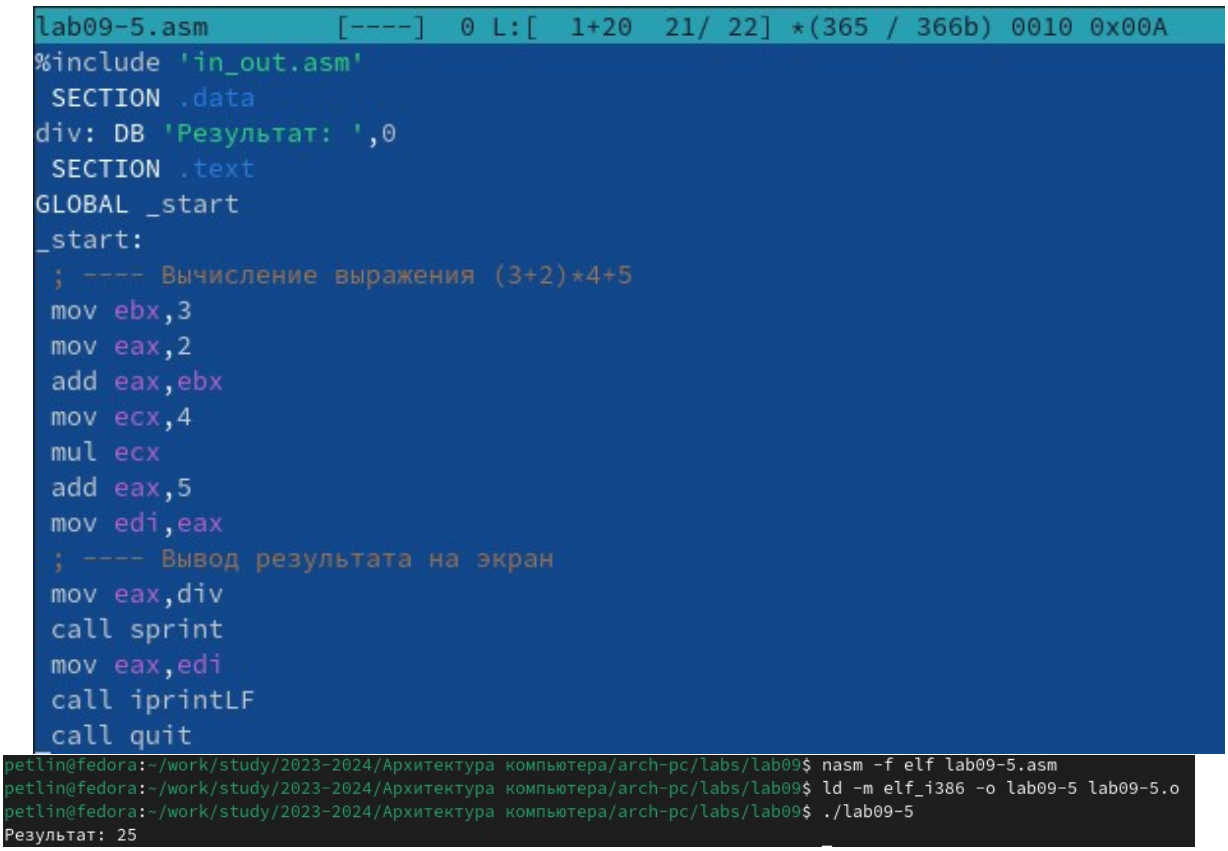


Создаём файл lab09-5.asm, в который вписываем текст программы из листинга 9.3. Создаем исполняемый файл и проверяем его работу. Программа работает неверно.





Создаем исполняемый файл и запускаем его в оболочке GDB. Смотрим на изменение значение регистров с помощью команды si.



Изменяем текст программы для корректной работы. Создаем исполняемый файл, который теперь работает верно.

5 Выводы

Мы приобрели навыки написания программ с использованием подпрограмм. Мы познакомились с методами отладки при помощи GDB и 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 c. — (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 c. — ISBN 978-1491941591.

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

8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 c. — 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. — БХВПетербург, 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).