

Лабораторная работа №9

. Понятие подпрограммы. Отладчик GDB.

Богданюк Анна Васильевна

Содержание

1	Цель работы	6
2	Задание	7
3	Теоретическое введение	8
4	Выполнение лабораторной работы	10
5	Выводы	33
	Список литературы	34

Список таблиц

Список иллюстраций

4.1	Создание каталога и файла	10
4.2	Файл lab09-1.asm	10
4.3	Создание и вывод	12
4.4	Создание и вывод	12
4.5	Файл lab09-2	14
4.6	Создание исполняемого файла	15
4.7	Загрузка в gdb	16
4.8	Проверка	16
4.9	Устанавливаю breakpoint и запуск	16
4.10	Дисассимилированный код	17
4.11	Переключение на отображение команд с Intel'овским синтаксисом	18
4.12	Режим псевдографики	18
4.13	info breakpoints	19
4.14	Устанавливаю точку останова	19
4.15	Точки останова	19
4.16	Содержимое регистров	20
4.17	Значение msg1	20
4.18	Изменение символа и вывод	20
4.19	msg2	21
4.20	Изменение значения регистра	21
4.21	Изменение значения регистра	22
4.22	Завершаю выполнение	22
4.23	Выхожу из GDB	23
4.24	Копирую файл	23
4.25	Создание файла	23
4.26	Загружаю файл	23
4.27	Установка точки останова и запуск	24
4.28	Аргументов 3 + название = 4	24
4.29	Позиции стека	25
4.30	Файл lab09-4.asm	25
4.31	Создание и запуск	26
4.32	Создание файла	26
4.33	Загружаю файл в отладчик	26
4.34	Проверка работы программы	26
4.35	Устанавливаю breakpoint и запускаю	27
4.36	Дисассимилированный код	27
4.37	Отображение команд с Intel'овским синтаксисом	28

4.38 Загружаю файл в отладчик	28
4.39 Изменение регистров	29
4.40 Значение регистра ebx	29
4.41 Ошибка	30
4.42 Результат	30
4.43 Файл lab09-4.asm	31
4.44 Создание и запуск	31

1 Цель работы

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

2 Задание

1. Выполнение лабораторной работы
2. Задания для самостоятельной 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.

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 управ-

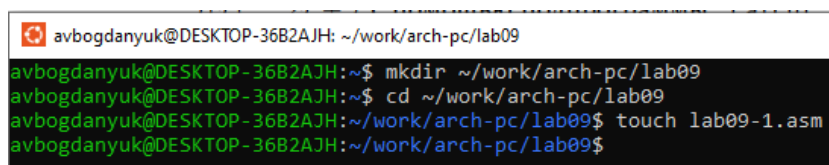
лять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам. Пошаговое выполнение — это выполнение программы 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.

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.

4 Выполнение лабораторной работы

1. Выполнение лабораторной работы

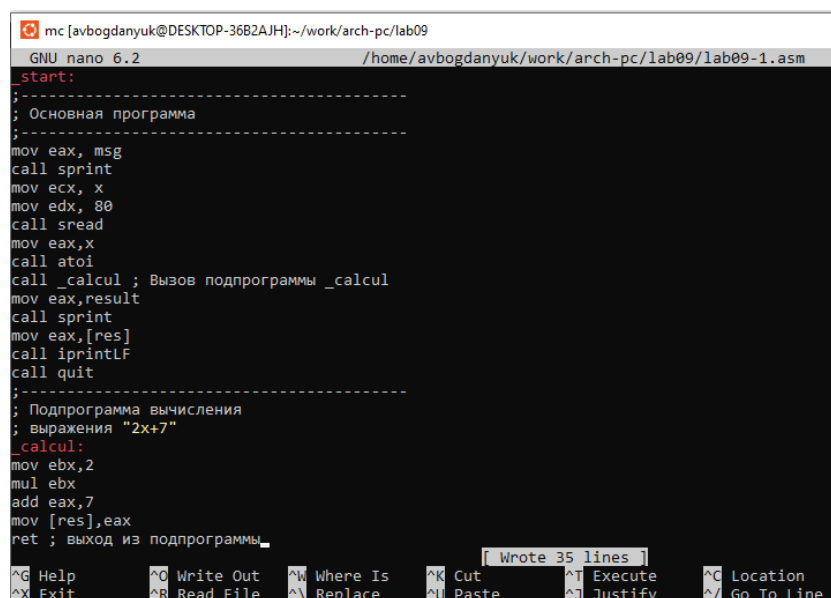
Создаю каталог для выполнения лабораторной работы № 9, перейдаю в него и создаю файл lab09-1.asm (рис. 4.1).



```
avbogdanyuk@DESKTOP-36B2AJH: ~/work/arch-pc/lab09
avbogdanyuk@DESKTOP-36B2AJH:~$ mkdir ~/work/arch-pc/lab09
avbogdanyuk@DESKTOP-36B2AJH:~$ cd ~/work/arch-pc/lab09
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ touch lab09-1.asm
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$
```

Рис. 4.1: Создание каталога и файла

Ввожу текст программы из листинга в файл lab09-1.asm (рис. 4.2).



```
mc [avbogdanyuk@DESKTOP-36B2AJH]:~/work/arch-pc/lab09
GNU nano 6.2 /home/avbogdanyuk/work/arch-pc/lab09/lab09-1.asm
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
[ Wrote 35 lines ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit      ^R Read File  ^N Replace    ^U Paste      ^J Justify    ^_ Go To Line
```

Рис. 4.2: Файл lab09-1.asm

ЛИСТИНГ:

```
%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

```

Создаю исполняемый файл и запускаю его. Программа работает корректно (рис. 4.3).

```

avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
2x+7=13
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$

```

Рис. 4.3: Создание и вывод

Изменяю текст программы так, чтобы сначала считалась $3x-1$, затем результат этого уравнения был x в уравнении $2x+7$. В первом уравнении $x=3$, выражение равно 8, значит $f(x)=16+7=23$. Создаю исполняемый файл и запускаю его. Программа работает корректно (рис. 4.4).

```

avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
2x+7=23
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ _

```

Рис. 4.4: Создание и вывод

Листинг:

```

#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:
call _subcalcul
mov ebx,2

```

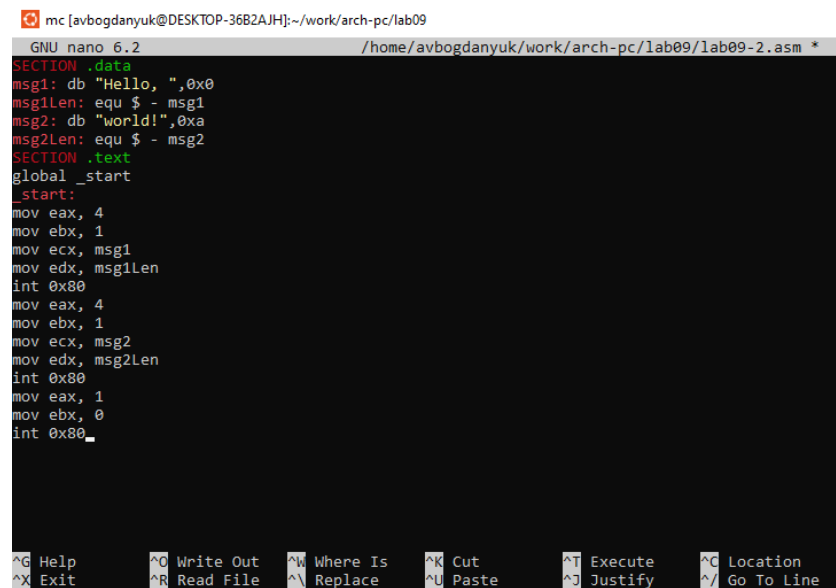
```

mul ebx
add eax, 7
mov [res], eax
ret

_subcalcul:
mov ebx, 3
mul ebx
add eax, -1
mov [res], eax
ret

```

Создаю файл lab09-2 и ввожу текст программы из листинга (рис. 4.5).



```

mc [avbogdanyuk@DESKTOP-36B2AJH]:~/work/arch-pc/lab09
GNU nano 6.2 /home/avbogdanyuk/work/arch-pc/lab09/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

```

Рис. 4.5: Файл lab09-2

Листинг:

```

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

```

Получаю исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’ (рис. 4.6).

```

avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$

```

Рис. 4.6: Создание исполняемого файла

Загружаю исполняемый файл в отладчик gdb (рис. 4.7).

```

avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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)

```

Рис. 4.7: Загрузка в gdb

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 4.8).

```

(gdb) run
Starting program: /home/avbogdanyuk/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 763) exited normally]
(gdb)

```

Рис. 4.8: Проверка

Для более подробного анализа программы устанавливаю брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запускаю её (рис. 4.9).

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/avbogdanyuk/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb)

```

Рис. 4.9: Устанавливаю breakpoint и запуск

Смотрю дисассимилированный код программы с помощью команды disassemble начиная с метки _start (рис. 4.10).


```

(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     $0x4,%eax
    0x08049005 <+5>:      mov     $0x1,%ebx
    0x0804900a <+10>:     mov     $0x804a000,%ecx
    0x0804900f <+15>:     mov     $0x8,%edx
    0x08049014 <+20>:     int     $0x80
    0x08049016 <+22>:     mov     $0x4,%eax
    0x0804901b <+27>:     mov     $0x1,%ebx
    0x08049020 <+32>:     mov     $0x804a008,%ecx
    0x08049025 <+37>:     mov     $0x7,%edx
    0x0804902a <+42>:     int     $0x80
    0x0804902c <+44>:     mov     $0x1,%eax
    0x08049031 <+49>:     mov     $0x0,%ebx
    0x08049036 <+54>:     int     $0x80
End of assembler dump.
(gdb)

```

Рис. 4.10: Дисассимилированный код

Переключая на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`. Существует два режима отображения синтаксиса машинных команд: режим Intel, используемый в том числе в NASM, и режим АТТ (значительно отличающийся внешне). По умолчанию в дизассемблере GDB принят режим АТТ. Переключиться на отображение команд с привычным Intel'овским синтаксисом можно, введя команду `set disassembly-flavor intel` (рис. 4.11).

```

(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. 4.11: Переключение на отображение команд с Intel'овским синтаксисом

Включаю режим псевдографики для более удобного анализа программы (рис. 4.12).

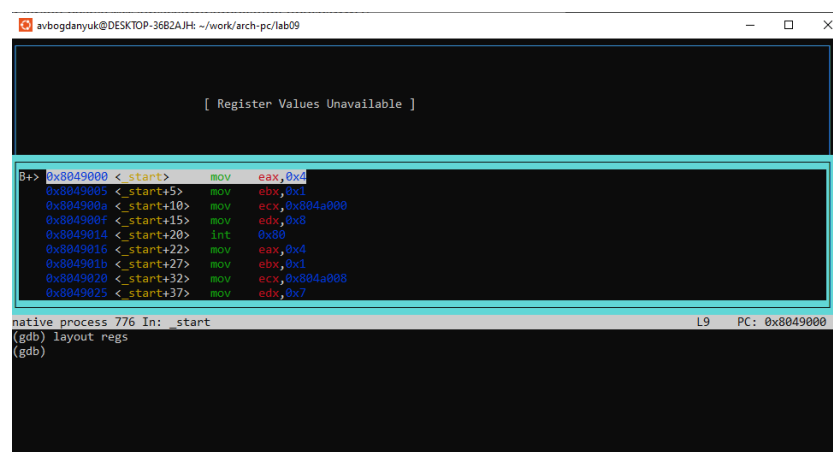


Рис. 4.12: Режим псевдографики

На предыдущих шагах была установлена точка останова по имени метки (_start).

Проверяю это с помощью команды `info breakpoints` (рис. 4.13).

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

Рис. 4.13: `info breakpoints`

Установим еще одну точку останова по адресу инструкции. Определяю адрес предпоследней инструкции (`mov ebx,0x0`) и устанавливаю точку останова (рис. 4.14).

```
b+ 0x8049031 <_start+49> mov ebx,0x0
native process 831 In: _start
(gdb) layout regs
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint      keep y 0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb)
```

Рис. 4.14: Устанавливаю точку останова

Смотрю информацию о всех установленных точках останова (рис. 4.15).

```
(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)
```

Рис. 4.15: Точки останова

Смотрю содержимое регистров с помощью команды `info registers` (рис. 4.16).

```

native process 831 In: _start
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
--Type <RET> for more, q to quit, c to continue without paging--eflags 0x202 [ IF ]
cs       0x23      35
ss       0x2b      43
ds       0x2b      43
es       0x2b      43
fs       0x0      0
gs       0x0      0
(gdb)

```

Рис. 4.16: Содержимое регистров

Смотрю значение переменной msg1 по имени. Значение переменной msg2 на рис.19 (рис. 4.17).

```

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

```

Рис. 4.17: Значение msg1

Изменяю первый символ переменной msg1 (рис. 4.18).

```

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

```

Рис. 4.18: Изменение символа и вывод

Вывожу содержимое переменной msg2. Изменяю первый символ на 'v' (рис. 4.19).

```

(gdb) x/1sb &msg2
0x804a008 <msg2>: "world!\n\034"
(gdb) set {char}&msg2='v'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "vor1d!\n\034"
(gdb)

```

Рис. 4.19: msg2

С помощью команды set изменяю значение регистра ebx (рис. 4.20).

```

avbogdanyuk@DESKTOP-3682AJH: ~/work/arch-pc/lab09
Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x32    50
esp      0xffffd1f0 0xffffd1f0  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

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 831 In: _start
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) x/1sb &msg2
0x804a008 <msg2>: "world!\n\034"
(gdb) set {char}&msg2='v'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "vor1d!\n\034"
(gdb) set $ebx=2
(gdb) p/s $ebx
$1 = 50
(gdb)

```

Рис. 4.20: Изменение значения регистра

С помощью команды set изменяю значение регистра ebx (рис. 4.21).

```

avbogdanyuk@DESKTOP-36B2AJHT: ~/work/arch-pc/lab09
Register group: general
eax    0x0      0      ecx    0x0      0
edx    0x0      0      ebx    0x2      2
esp    0xffffd1f0 0xffffd1f0  ebp    0x0      0x0
esi    0x0      0      edi    0x0      0
eip    0x8049000 0x8049000 <_start>  eflags 0x202     [ IF ]
cs     0x2b     43      ss     0x2b     43
ds     0x2b     43      es     0x2b     43

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 831 In: _start
(gdb) x/1sb &msg2
0x804a008 <msg2>: "world!\n\034"
(gdb) set {char}&msg2='v'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "vorld!\n\034"
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb)

```

Рис. 4.21: Изменение значения регистра

Завершаю выполнение программы с помощью команды continue (рис. 4.22).

```

avbogdanyuk@DESKTOP-36B2AJHT: ~/work/arch-pc/lab09
eax    0x0      0      ecx    0x0      0
edx    0x1      1      ebx    0x804a008 134520840
esi    0x7      7      ebp    0x1      0
esp    0x0      0      edi    0x0      0
eip    0x8049000 0x8049000 <_start>  eflags 0x202     [ IF ]
cs     0x2b     43      ss     0x2b     43
ds     0x2b     43      es     0x2b     43

0x8049025 <_start+37> int     0x80 7
0x804902a <_start+42> mov    eax,0x4
0x804902c <_start+44> mov    ebx,0x1
B> 0x8049031 <_start+49> mov    ebx,0x804a008
0x8049036 <_start+54> int     0x80
0x8049038 add    BYTE PTR [eax],a1
0x804903a add    BYTE PTR [eax],a1

native process 831 In: _start
(gdb) set {char}&msg2='v'
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb) c
Continuing.
hello, world!
Breakpoint 2, _start () at lab09-2.asm:20
(gdb)

```

Рис. 4.22: Завершаю выполнение

Выхожу из GDB с помощью команды quit (рис. 4.23).

```
(gdb) q
A debugging session is active.

    Inferior 1 [process 831] will be killed.

Quit anyway? (y or n) _
```

Рис. 4.23: Выхожу из GDB

Копируйте 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 (рис. 4.24).

```
avbogdanyuk@DESKTOP-3682AJH:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
avbogdanyuk@DESKTOP-3682AJH:~/work/arch-pc/lab09$
```

Рис. 4.24: Копирую файл

Создаю исполняемый файл (рис. 4.25).

```
avbogdanyuk@DESKTOP-3682AJH:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
avbogdanyuk@DESKTOP-3682AJH:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
avbogdanyuk@DESKTOP-3682AJH:~/work/arch-pc/lab09$
```

Рис. 4.25: Создание файла

Для загрузки в gdb программы с аргументами необходимо использовать ключ `-args`. Загружаю исполняемый файл в отладчик, указав аргументы (рис. 4.26).

```
avbogdanyuk@DESKTOP-3682AJH:~/work/arch-pc/lab09$ gdb --args lab09-3 2 3 '4'
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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)
```

Рис. 4.26: Загружаю файл

Для начала устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. 4.27).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/avbogdanyuk/work/arch-pc/lab09/lab09-3 2 3 4

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 4.27: Установка точки останова и запуск

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы) (рис. 4.28).

```
(gdb) x/x $esp
0xffffd1e0: 0x00000004
(gdb)
```

Рис. 4.28: Аргументов 3 + название = 4

Смотрю остальные позиции стека – по адресу [esp+4] располагается адрес в памяти где находится имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д. (рис. 4.29).


```


(gdb) x/s *(void**)(esp + 4)
0xffffd341:    "/home/avbogdanyuk/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd36e:    "2"
(gdb) x/s *(void**)(esp + 12)
0xffffd370:    "3"
(gdb) x/s *(void**)(esp + 16)
0xffffd372:    "4"
(gdb) x/s *(void**)(esp + 20)
0x0:    <error: Cannot access memory at address 0x0>
(gdb) x/s *(void**)(esp + 24)
0xffffd374:    "SHELL=/bin/bash"
(gdb)

```

Рис. 4.29: Позиции стека

2. Задания для самостоятельной работы

Создаю файл lab09-4.asm и ввожу в него текст программы листинга (рис. 4.30).



```

mc [avbogdanyuk@DESKTOP-36B2AJH]:~/work/arch-pc/lab09
GNU nano 6.2 /home/avbogdanyuk/work/arch-pc/lab09/lab09-4.asm *
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
File Name to Write: /home/avbogdanyuk/work/arch-pc/lab09/lab09-4.asm
^G Help      M-D DOS Format  M-A Append    M-
^C Cancel    M-M Mac Format  M-P Prepend

```

Рис. 4.30: Файл lab09-4.asm

Создаю исполняемый файл и запускаю его. Действительно, программа работает неверно (рис. 4.31).

```

avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ ./lab09-4
Результат: 10
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$

```

Рис. 4.31: Создание и запуск

Создаю исполняемый файл для работы с GDB (рис. 4.32).

```

avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-4.lst lab09-4.asm
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$

```

Рис. 4.32: Создание файла

Загружаю исполняемый файл в отладчик gdb (рис. 4.33).

```

avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ gdb lab09-4
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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 lab09-4...
(gdb)

```

Рис. 4.33: Загружаю файл в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 4.34).

```

(gdb) run
Starting program: /home/avbogdanyuk/work/arch-pc/lab09/lab09-4
Результат: 10
[Inferior 1 (process 1415) exited normally]
(gdb)

```

Рис. 4.34: Проверка работы программы

Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запускаю её. (рис. 4.35).

```
(gdb) break _start
Breakpoint 1 at 0x80490e8: file lab09-4.asm, line 8.
(gdb) run
Starting program: /home/avbogdanyuk/work/arch-pc/lab09/lab09-4

Breakpoint 1, _start () at lab09-4.asm:8
8      mov ebx,3
(gdb)
```

Рис. 4.35: Устанавливаю breakpoint и запускаю

Смотрю дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 4.36).

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

Рис. 4.36: Дисассимилированный код

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 4.37).

```

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

```

Рис. 4.37: Отображение команд с Intel'овским синтаксисом

На предыдущих шагах была установлена точка останова по имени метки (_start).
Проверяю это с помощью команды info breakpoints (рис. 4.38).

```

native process 1425 In: _start
(gdb) layout regs
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint       keep y 0x080490e8 lab09-4.asm:8
          breakpoint already hit 1 time
(gdb)

```

Рис. 4.38: Загружаю файл в отладчик

С помощью `stepi` буду отслеживать, как изменяются значения регистров (рис. 4.39).

```

avbogdanyuk@DESKTOP-36B2AJH: ~/work/arch-pc/lab09
--Register group: general--
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x3      3
esp      0xffffd1e0 0xffffd1e0  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x80490ed 0x80490ed < _start+5>  eflags   0x202    [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43

B+ 0x80490e8 < _start>    mov     ebx,0x3
B+ 0x80490ed < _start+5>  mov     eax,0x2
B+ 0x80490f2 < _start+10> add     ebx,eax
0x80490f4 < _start+12>    mov     ecx,0x4
0x80490f9 < _start+17>    mul     ecx
0x80490fb < _start+19>    add     ebx,0x5
0x80490fe < _start+22>    mov     edi,ebx
0x8049100 < _start+24>    mov     eax,0x804a000
0x8049105 < _start+29>    call    0x804900f <sprint>

native process 2385 In: _start
cs      0x23     35
ss      0x2b     43
ds      0x2b     43
es      0x2b     43
fs      0x0      0
gs      0x0      0
(gdb) stepi
Breakpoint 2, 0x80490ed in _start ()
(gdb)

```

Рис. 4.39: Изменение регистров

На 3 шаге значение `ebx` = 5 (рис. 4.40).

```

avbogdanyuk@DESKTOP-36B2AJH: ~/work/arch-pc/lab09
--Register group: general--
eax      0x2      2      ecx      0x0      0
edx      0x0      0      ebx      0x5      5
esp      0xffffd1e0 0xffffd1e0  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x80490f4 0x80490f4 < _start+12>  eflags   0x206    [ PF IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43

B+ 0x80490e8 < _start>    mov     ebx,0x3
B+ 0x80490ed < _start+5>  mov     eax,0x2
B+ 0x80490f2 < _start+10> add     ebx,eax
> 0x80490f4 < _start+12>  mov     ecx,0x4
0x80490f9 < _start+17>    mul     ecx
0x80490fb < _start+19>    add     ebx,0x5
0x80490fe < _start+22>    mov     edi,ebx
0x8049100 < _start+24>    mov     eax,0x804a000
0x8049105 < _start+29>    call    0x804900f <sprint>

native process 2385 In: _start
Breakpoint 2, 0x80490ed in _start ()
(gdb) уеуэш
Undefined command: "". Try "help".
(gdb) stepi
Breakpoint 3, 0x80490f2 in _start ()
(gdb) stepi
0x80490f4 in _start ()
(gdb)

```

Рис. 4.40: Значение регистра `ebx`

На шаге 5 произошла ошибка. `esx` было умножено не на `ebx`, а на `eax`. Значение `ebx` а этом шаге должно было быть равно 20, но оно равно $2 \cdot 4 = 8$ (рис. 4.41).

```

avbogdanyuk@DESKTOP-36B2AJH: ~/work/arch-pc/lab09
Register group: general
eax      0x0      8      ecx      0x4      4
edx      0x0      0      ebx      0x5      5
esp      0xffffd1e0 0xffffd1e0  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x80490fb 0x80490fb < start+19>  eflags   0x202    [ IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43

B+ 0x80490e8 < _start>    mov     ebx,0x3
B+ 0x80490ed < _start+5>  mov     eax,0x2
B+ 0x80490f2 < _start+10> add     ebx,eax
0x80490f4 < _start+12>    mov     ecx,0x4
0x80490f9 < _start+17>    mul     ecx
> 0x80490fb < _start+19>  add     ebx,0x5
0x80490fe < _start+22>    mov     edi,ebx
0x8049100 < _start+24>    mov     eax,0x804a000
0x8049105 < _start+29>    call    0x804900f <sprint>

native process 2385 In: _start
(gdb) stepi
Breakpoint 3, 0x80490f2 in _start ()
(gdb) stepi
0x80490f4 in _start ()
(gdb) stepi
0x80490f9 in _start ()
(gdb) stepi
0x80490fb in _start ()
(gdb)

```

Рис. 4.41: Ошибка

На этом шаге виден результат вычисления программы. $ebx = ebx + 5$, но, так как $ebx = 5 = 3+2$, а не $ebx=20+5$, конечный результат равен не 25, а 10 (рис. 4.42).

```

avbogdanyuk@DESKTOP-36B2AJH: ~/work/arch-pc/lab09
Register group: general
eax      0x0      8      ecx      0x4      4
edx      0x0      0      ebx      0xa     10
esp      0xffffd1e0 0xffffd1e0  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x80490fe 0x80490fe < _start+22>  eflags   0x206    [ PF IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43

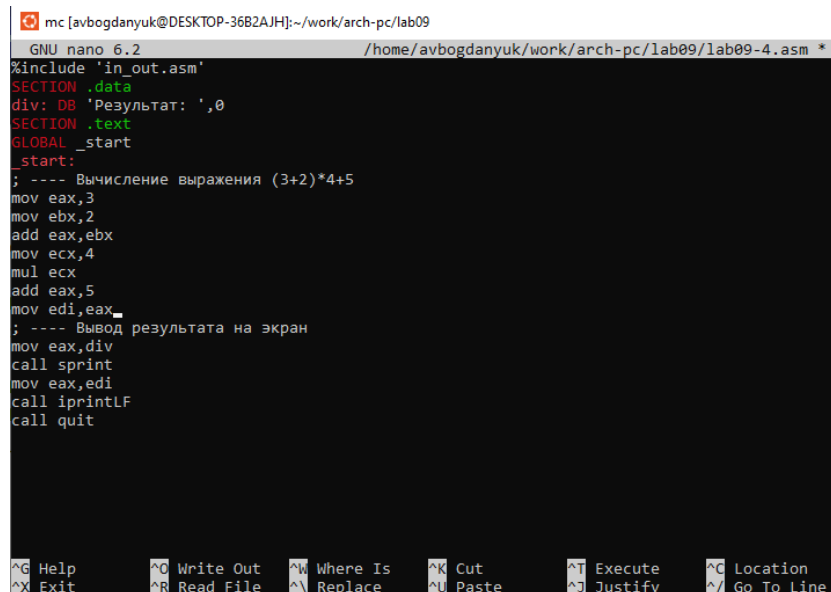
B+ 0x80490e8 < _start>    mov     ebx,0x3
B+ 0x80490ed < _start+5>  mov     eax,0x2
B+ 0x80490f2 < _start+10> add     ebx,eax
0x80490f4 < _start+12>    mov     ecx,0x4
0x80490f9 < _start+17>    mul     ecx
0x80490fb < _start+19>  add     ebx,0x5
> 0x80490fe < _start+22>  mov     edi,ebx
0x8049100 < _start+24>    mov     eax,0x804a000
0x8049105 < _start+29>    call    0x804900f <sprint>

native process 2385 In: _start
(gdb) stepi
Breakpoint 3, 0x80490f2 in _start ()
(gdb) stepi
0x80490f4 in _start ()
(gdb) stepi
0x80490f9 in _start ()
(gdb) stepi
0x80490fb in _start ()
(gdb) stepi
0x80490fe in _start ()
(gdb)

```

Рис. 4.42: Результат

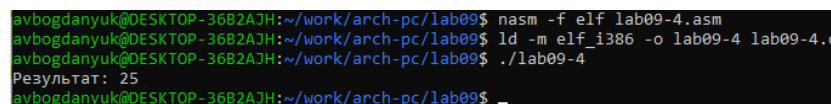
Изменяю текст программы так, чтобы вычисления были верными (рис. 4.43).



```
mc [avbogdanyuk@DESKTOP-36B2AJH]:~/work/arch-pc/lab09
GNU nano 6.2 /home/avbogdanyuk/work/arch-pc/lab09/lab09-4.asm *
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov eax,3
mov ebx,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
^G Help      ^O Write Out  ^M Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit      ^R Read File  ^I Replace    ^U Paste      ^J Justify    ^_ Go To Line
```

Рис. 4.43: Файл lab09-4.asm

Создаю исполняемый файл и запускаю его. Программа работает корректно (рис. 4.44).



```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$ ./lab09-4
Результат: 25
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab09$
```

Рис. 4.44: Создание и запуск

Листинг:

```
%include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov eax,3
```

```
mov ebx,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```


5 Выводы

В ходе выполнения лабораторной работы были приобретены навыки написания программ с использованием подпрограмм. Я ознакомилась с методами отладки при помощи GDB и его основными возможностями.

Список литературы