

Отчет по лабораторной работе №10

Архитектура компьютера

Попова Елизавета Сергеевна

Содержание

1	Цель работы	6
2	Задание	7
3	Выполнение лабораторной работы	8
4	Задание для самостоятельной работы	24
5	Выводы	38

Список иллюстраций

3.1	8
3.2	9
3.3	10
3.4	11
3.5	12
3.6	12
3.7	12
3.8	13
3.9	14
3.10	14
3.11	15
3.12	15
3.13	16
3.14	17
3.15	17
3.16	17
3.17	18
3.18	18
3.19	19
3.20	19
3.21	19
3.22	19
3.23	20
3.24	20
3.25	21
3.26	21
3.27	21
3.28	21
3.29	22
3.30	22
3.31	22
3.32	23
4.1	25
4.2	26
4.3	26
4.4	26

4.5	27
4.6	27
4.7	27
4.8	27
4.9	27
4.10	28
4.11	29
4.12	30
4.13	31
4.14	32
4.15	33
4.16	34
4.17	35
4.18	36
4.19	37
4.20	37

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. Реализовать подпрограммы в NASM.
2. Выполнить отладку программ с помощью GDB.
3. Отработать добавление точек останова.
4. Поработа с данными программы в GDB.
5. Отработать обработку аргументов командной строки в GDB.
6. Выполнить задание для самостоятельной работы.

3 Выполнение лабораторной работы

Создаем каталог для выполнения лабораторной работы № 10, переходим в него и создаем файл lab10-1.asm: (рис. 3.1)

```
[laisacreate@10 ~]$ mkdir work/arch-pc/lab10
[laisacreate@10 ~]$ cd work/arch-pc/lab10
[laisacreate@10 lab10]$ touch lab10-1.asm
[laisacreate@10 lab10]$ ls
in_out.asm lab10-1.asm
[laisacreate@10 lab10]$
```

Рис. 3.1: .

В качестве примера рассмотрим программу вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы `_calcul`. В данном примере `x` вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучим текст программы (Листинг 10.1). Введем в файл `lab10-1.asm` текст программы из листинга 10.1 (рис. 3.2). Создадим исполняемый файл и проверим его работу (рис. 3.3).


```

%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
rezs: 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 [rez],eax
ret ; выход из подпрограммы

```

Рис. 3.2: .

```
[laisacrete@10 lab10]$ nasm -f elf lab10-1.asm
[laisacrete@10 lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[laisacrete@10 lab10]$ ./lab10-1
Введите x: 3
2x+7=13
```

Рис. 3.3: .

Изменим текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. Т.е. x передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение $g(x)$, результат возвращается в `_calcul` и 8 вычисляется выражение $f(g(x))$. Результат возвращается в основную программу для вывода результата на экран (рис. 3.4),(рис. 3.5).

```

%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
rezs: 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,[rezs]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx,2
mul ebx
add eax,7
mov [rezs],eax
ret ; выход из подпрограммы

_subcalcul:
mov ebx,3
mul ebx
sub eax,1
mov [rezs],eax
ret

```

Рис. 3.4: .

```
[laisacrete@10 lab10]$ nasm -f elf lab10-1.asm
[laisacrete@10 lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[laisacrete@10 lab10]$ ./lab10-1
Введите x: 3
2x+7=23
```

Рис. 3.5: .

Создадим файл lab10-2.asm с текстом программы из Листинга 10.2. (Программа печати сообщения Hello world!): (рис. 3.6),(рис. 3.7).

```
[laisacrete@10 lab10]$ touch lab10-2.asm
```

Рис. 3.6: .

```
[laisacrete@10 lab10]$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
[laisacrete@10 lab10]$ ld -m elf_i386 -o lab10-2 lab10-2.o
[laisacrete@10 lab10]$ gdb lab10-2
```

Рис. 3.7: .

Получаем исполняемый файл. Для работы с GDB в исполняемый файл добавляем отладочную информацию, для этого трансляцию программ провели с ключом '-g'. Загружаем исполняемый файл в отладчик gdb (рис. 3.8).

```

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. 3.8: .

Загружаем исполняемый файл в отладчик gdb. Проверяем работу программы, запустив ее в оболочке GDB с помощью команды run: (рис. 3.9).

```
(gdb) r
Starting program: /home/laisacrete/work/arch-pc/lab10/lab10-2

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 separate debug info for /home/laisacrete/work/arch-pc/lab10/system-supplied DSO at 0xf7ffc000...
Hello, world!
[Inferior 1 (process 3310) exited normally]
```

Рис. 3.9: .

Для более подробного анализа программы установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её (рис. 3.10).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 9.
(gdb) run
Starting program: /home/laisacrete/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:9
9      mov eax, 4
```

Рис. 3.10: .

Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 3.11).

```

(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.

```

Рис. 3.11: .

Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`. (рис. 3.12)

```

(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.

```

Рис. 3.12: .

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel: в

АТТ перед адресом регистра ставится \$, а перед названием регистра %, сначала записывается адрес, а потом - регистр. В Intel сначала регистр, а потом адрес, и перед ними ничего не ставится. Включим режим псевдографики для более удобного анализа программы (рис. 3.13).

The screenshot shows a debugger window with a dark background. At the top, a message box says "[Register Values Unavailable]". Below it, a list of assembly instructions is displayed in a table-like format. The first instruction is highlighted with a blue background. At the bottom, there is a command prompt showing the execution of GDB commands.

Address	Disassembly	Comment
0x8049000	mov	eax,0x4
0x8049005	mov	ebx,0x1
0x804900a	mov	ecx,0x804a000
0x804900f	mov	edx,0x8
0x8049014	int	0x80
0x8049016	mov	eax,0x4
0x804901b	mov	ebx,0x1
0x8049020	mov	ecx,0x804a008
0x8049025	mov	edx,0x7

```

native process 3346 In: _start
(gdb) layout regs
(gdb)
  
```

Рис. 3.13: .

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать или как номер строки программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»: На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверим это с помощью команды `info breakpoints` (кратко `i b`). Установим еще одну точку останова по адресу инструкции. Адрес инструкции увидим в средней части экрана в левом столбце соответствующей инструкции. Определим адрес предпоследней инструкции (`mov ebx,0x0`) и установим точку останова (рис. 3.14).


```
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab10-2.asm, line 20.
```

Рис. 3.14: .

Посмотрим информацию о всех установленных точках останова: (рис. 3.15)

```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab10-2.asm:9
         breakpoint already hit 1 time
2        breakpoint     keep y   0x08049031 lab10-2.asm:20
```

Рис. 3.15: .

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполнили 5 инструкций с помощью команды stepi (или si) и проследили за изменением значений регистров (рис. 3.16), (рис. 3.17).

```
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
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
```

Рис. 3.16: .

```

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
    0x804902c <_start+44> mov    eax,0x1

native process 3346 In: _start L14 PC: 0:
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--cfs 0x0 0
gs       0x0      0
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi

```

Рис. 3.17: .

Изменяются значения регистров: eax, ecx, edx, ebx. Посмотрим содержимое регистров с помощью команды info registers (или i r) (рис. 3.18).

```

eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1f0 0xffffd1f0
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

```

Рис. 3.18: .

Для отображения содержимого памяти можно использовать команду x, которая выдаёт содержимое ячейки памяти по указанному адресу. Формат, в котором выводятся данные, можно задать после имени команды через косую черту: x/NFU. С помощью команды x & также можно посмотреть содержимое переменной. Посмотрим значение переменной msg1 по имени. (рис. 3.19)

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

Рис. 3.19: .

Посмотрим значение переменной msg2 по адресу. Адрес переменной определим по дизассемблированной инструкции. Посмотрим инструкцию mov esx,msg2 которая записывает в регистр esx адрес переменной msg2 (рис. 3.20).

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

Рис. 3.20: .

Изменим 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. Изменим первый символ переменной msg1 (рис. 3.21)

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

Рис. 3.21: .

Замените первый символ во второй переменной msg2 (рис. 3.22).

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

Рис. 3.22: .

Чтобы посмотреть значения регистров используется команда print /F. Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. 3.23).

```
(gdb) p/s $edx
$1 = 8
(gdb) p/x $edx
$2 = 0x8
(gdb) p/t $edx
$3 = 1000
(gdb) p/s $edx
$4 = 8
(gdb) set $ebx='2'
(gdb) p/s $ebx
$5 = 50
```

Рис. 3.23: .

С помощью команды set изменим значение регистра ebx. Разница вывода команд p/s \$ebx: (рис. 3.24).

```
(gdb) set $ebx=2
(gdb) p/s $ebx
$6 = 2
```

Рис. 3.24: .

Завершим выполнение программы с помощью команды continue (сокращенно c) и выйдем из GDB с помощью команды quit (сокращенно q) (рис. 3.25), (рис.

3.26).

```
(gdb) c
Continuing.
Rorld!

Breakpoint 2, _start () at lab10-2.asm:20
(gdb) c
Continuing.
[Inferior 1 (process 3346) exited normally]
```

Рис. 3.25: .



Рис. 3.26: .

Скопируем файл lab9-2.asm, созданный при выполнении лабораторной работы №9, с программой выводящей на экран аргументы командной строки (Листинг 9.2) в файл с именем lab10-3.asm: (рис. 3.27)

```
[laisacrete@10 lab10]$ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
```

Рис. 3.27: .

Создадим исполняемый файл (рис. 3.28).

```
[laisacrete@10 lab10]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
[laisacrete@10 lab10]$ ld -m elf_i386 -o lab10-3 lab10-3.o
```

Рис. 3.28: .

Для загрузки в gdb программы с аргументами используем ключ--args. Загрузим исполняемый файл в отладчик, указав аргументы:(рис. 3.29).

```
[laisacrete@i10 lab10]$ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
```

Рис. 3.29: .

Как отмечалось в предыдущей лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Исследуем расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установим точку старта перед первой инструкцией в программе и запустим ее.(рис. 3.30)

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 5.
(gdb) run
Starting program: /home/laisacrete/work/arch-pc/lab10/lab10-3 аргумент1 аргумент 2 аргумент\ 3

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.

Breakpoint 1, _start () at lab10-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
```

Рис. 3.30: .

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы):(рис. 3.31)

```
(gdb) x/x $esp
0xffffd1a0: 0x00000005
```

Рис. 3.31: .


Как видно, число аргументов равно 5 – это имя программы lab10-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Посмотрим остальные позиции стека – по адресу [esp+4] располагается адрес в памяти где находится имя программы, по адресу [esp+8] хранится адрес первого аргумента, по адресу [esp+12] – второго и т.д. (рис. 3.32)

```
(gdb) x/s *(void**)(esp + 4)
0xffffd358:    "/home/laisacreate/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd385:    "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd397:    "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd3a8:    "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd3aa:    "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
```

Рис. 3.32: .

4 Задание для самостоятельной работы

Преобразуем программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму (рис. 4.1), (рис. 4.2).

Открыть ▾ 

lab10-4.asm
~/work/arch-pc/lab10

```
%include 'in_out.asm'

SECTION .data
msg db "Результат: ",0

SECTION .text
GLOBAL _start

_start:
pop ecx

pop edx

sub ecx,1

mov esi,0

next:
cmp ecx,0h
jz_end

pop eax
call atoi

call _func

loop next
_end:
mov eax,msg
call sprint
mov eax,esi
call iprintLF
call quit

_func:
    mov eax, eax
    mov ebx,6
    mul ebx
    add eax,13
    add esi,eax
    ret
```

Рис. 4.1: .

```

[laissacrete@10 lab10]$ nasm -f elf lab10-4.asm
lab10-4.asm:20: warning: label alone on a line without a colon
[-w+label-orphan]
[laissacrete@10 lab10]$ ld -m elf_i386 -o lab10-4 lab10-4.o
[laissacrete@10 lab10]$ ./lab10-4
Ошибка сегментирования (стек памяти сброшен на диск)
[laissacrete@10 lab10]$ ./lab10-4 0
Результат: 13
[laissacrete@10 lab10]$ ./lab10-4 1
Результат: 19

```

Рис. 4.2: .

В листинге 10.3 приведена программа вычисления выражения $(3 + 2) * 4 + 5$. Создаем файл (рис. 4.3), запишем код листинга (рис. 4.4), создадим исполняющий файл (рис. 4.5), при запуске обнаружим вывод неверного результата (рис. 4.6).

```

[laissacrete@10 lab10]$ touch lab10-5.asm

```

Рис. 4.3: .

```

Открыть ▾ + lab10-5.asm
~/work/arch-pc/lab10

%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

```

Рис. 4.4: .

```

[laisacreate@10 lab10]$ nasm -f elf -g -l lab10-5.lst lab10-5.asm
[laisacreate@10 lab10]$ ld -m elf_i386 -o lab10-5 lab10-5.o

```

Рис. 4.5: .

```

[laisacreate@10 lab10]$ ./lab10-5
Результат: 10

```

Рис. 4.6: .

Запустим файл в отладчике GDB (рис. 4.7), установим точку останова (рис. 4.8), запустим код (рис. 4.9), включим режим псевдографики (рис. 4.10), пошагово пройдем все строчки кода (рис. 4.11), (рис. 4.12), (рис. 4.13), (рис. 4.14), (рис. 4.15), (рис. 4.16), (рис. 4.17), (рис. 4.18). Обнаружили ошибку: вместо регистра `ebx` на 4 умножался `eax`, а 5 прибавлялась не к произведению, а только к `ebx`, исправим её (рис. 4.19), проверим результат работы программы (рис. 4.20).

```

[laisacreate@10 lab10]$ gdb lab10-5
GNU gdb (GDB) Fedora 11.2-3.fc36

```

Рис. 4.7: .

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-5.asm, line 8.

```

Рис. 4.8: .

```

(gdb) r
Starting program: /home/laisacreate/work/arch-pc/lab10/lab10-5

```

Рис. 4.9: .

```
[ Register Values Unavailable ]

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
      0x80490fb <_start+19> add    $0x5,%ebx
      0x80490fe <_start+22> mov    %ebx,%edi
      0x8049100 <_start+24> mov    $0x804a000,%eax
      0x8049105 <_start+29> call   0x804900f <sprint>
      0x804910a <_start+34> mov    %edi,%eax
      0x804910c <_start+36> call   0x8049086 <iprintf>

native process 4432 In: _start
(gdb) layout regs
(gdb) █
```

Рис. 4.10: .

```
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 ]

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
0x80490fb <_start+19>   add    $0x5,%ebx
0x80490fe <_start+22>   mov    %ebx,%edi
0x8049100 <_start+24>   mov    $0x804a000,%eax
0x8049105 <_start+29>   call  0x804900f <sprint>
0x804910a <_start+34>   mov    %edi,%eax
0x804910c <_start+36>   call  0x8049086 <iprintLF>

native process 4432 In: _start
(gdb) layout regs
(gdb) stepi
(gdb)
```

Рис. 4.11:.

```
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0
esi      0x0      0
edi      0x0      0
eip      0x80490f2 0x80490f2 <_start+10>
eflags   0x202    [ IF ]

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
   0x80490fb <_start+19> add    $0x5,%ebx
   0x80490fe <_start+22> mov    %ebx,%edi
   0x8049100 <_start+24> mov    $0x804a000,%eax
   0x8049105 <_start+29> call   0x804900f <sprint>
   0x804910a <_start+34> mov    %edi,%eax
   0x804910c <_start+36> call   0x8049086 <iprintLF>

native process 4432 In: _start
(gdb) layout regs
(gdb) stepi
(gdb) stepi
(gdb)
```

Рис. 4.12: .

```

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 ]

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
0x80490fb <_start+19>   add    $0x5,%ebx
0x80490fe <_start+22>   mov    %ebx,%edi
0x8049100 <_start+24>   mov    $0x804a000,%eax
0x8049105 <_start+29>   call   0x804900f <sprint>
0x804910a <_start+34>   mov    %edi,%eax
0x804910c <_start+36>   call   0x8049086 <iprintLF>

native process 4432 In: _start
(gdb) layout regs
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 4.13: .

```

eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f9 0x80490f9 <_start+17>
eflags   0x206    [ PF IF ]

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
    0x80490fb <_start+19> add    $0x5,%ebx
    0x80490fe <_start+22> mov    %ebx,%edi
    0x8049100 <_start+24> mov    $0x804a000,%eax
    0x8049105 <_start+29> call   0x804900f <sprint>
    0x804910a <_start+34> mov    %edi,%eax
    0x804910c <_start+36> call   0x8049086 <iprintLF>

native process 4432 In: _start
(gdb) layout regs
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 4.14: .


```

eax      0x8      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 ]

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
> 0x80490fb <_start+19> add    $0x5,%ebx
    0x80490fe <_start+22> mov    %ebx,%edi
    0x8049100 <_start+24> mov    $0x804a000,%eax
    0x8049105 <_start+29> call   0x804900f <sprint>
    0x804910a <_start+34> mov    %edi,%eax
    0x804910c <_start+36> call   0x8049086 <iprintLF>

native process 4432 In: _start
(gdb) layout regs
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 4.15:.

eax	0x8	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]


```

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
    0x80490fb <_start+19>   add     $0x5,%ebx
> 0x80490fe <_start+22>   mov     %ebx,%edi
    0x8049100 <_start+24>   mov     $0x804a000,%eax
    0x8049105 <_start+29>   call    0x804900f <sprint>
    0x804910a <_start+34>   mov     %edi,%eax
    0x804910c <_start+36>   call    0x8049086 <iprintLF>

```



```

native process 4432 In: _start
(gdb) layout regs
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 4.16: .

```
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0xa      10
eip      0x8049100 0x8049100 <_start+24>
eflags   0x206    [ PF IF ]

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
    0x80490fb <_start+19> add    $0x5,%ebx
    0x80490fe <_start+22> mov    %ebx,%edi
> 0x8049100 <_start+24> mov    $0x804a000,%eax
    0x8049105 <_start+29> call   0x804900f <sprint>
    0x804910a <_start+34> mov    %edi,%eax
    0x804910c <_start+36> call   0x8049086 <iprintf>

native process 4432 In: _start
(gdb) layout regs
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)
```

Рис. 4.17:.

```

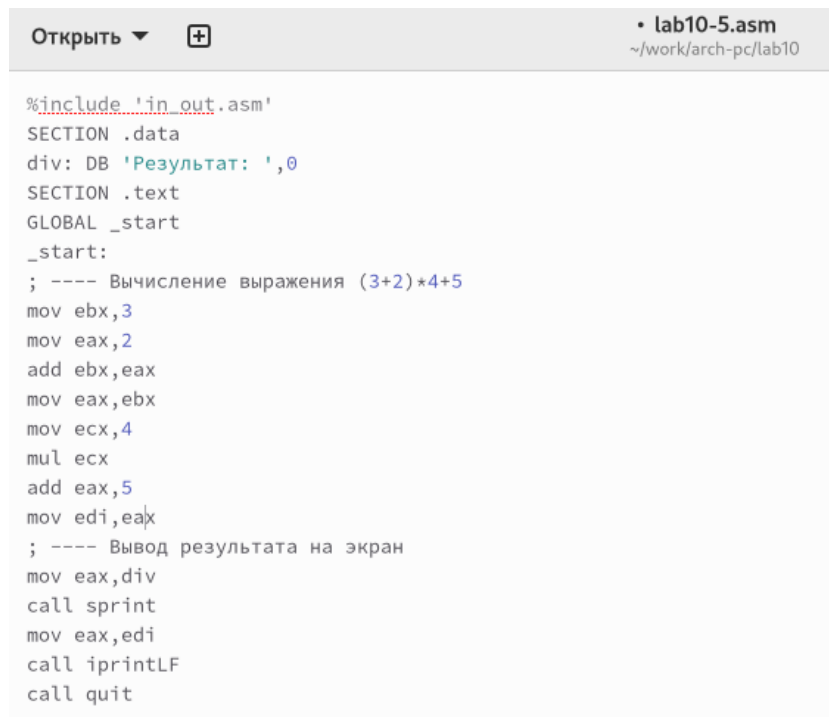
eax      0x804a000      134520832
ecx      0x4            4
edx      0x0            0
ebx      0xa            10
esp      0xffffd1e0     0xffffd1e0
ebp      0x0            0
esi      0x0            0
edi      0xa            10
eip      0x8049105      0x8049105 <_start+29>
eflags   0x206          [ PF IF ]

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
    0x80490fb <_start+19> add     $0x5,%ebx
    0x80490fe <_start+22> mov     %ebx,%edi
    0x8049100 <_start+24> mov     $0x804a000,%eax
> 0x8049105 <_start+29> call    0x804900f <sprint>
    0x804910a <_start+34> mov     %edi,%eax
    0x804910c <_start+36> call    0x8049086 <iprintLF>

native process 4432 In: _start
(gdb) layout regs
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 4.18: .




```
Открыть ▾  • lab10-5.asm  
~/work/arch-pc/lab10  
  
%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 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
```

Рис. 4.19: .

```
[laisacrete@10 lab10]$ nasm -f elf -g -l lab10-5.lst lab10-5.asm  
[laisacrete@10 lab10]$ ld -m elf_i386 -o lab10-5 lab10-5.o  
[laisacrete@10 lab10]$ ./lab10-5  
Результат: 25
```

Рис. 4.20: .

5 Выводы

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