

Отчет по лабораторной работе №9

Дисциплина: Архитектура компьютера

Челухаев Кирилл Александрович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Реализация подпрограмм в NASM	8
4.2	Отладка программ с помощью GDB	10
4.3	Задание для самостоятельной работы	17
5	Выводы	22
	Список литературы	23

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. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Задание 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.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

Я создал каталог для выполнения лабораторной работы № 9, перешел в него и создал файл lab9-1.asm (рис. ??).

```
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc $ mkdir lab09
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc $ cd lab09
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09 $ touch lab9-1.asm
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09 $
```

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.

```
%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

```

Первые строки программы отвечают за вывод сообщения на экран (call sprint), чтение данных введенных с клавиатуры (call sread) и преобразования введенных данных из символьного вида в численный (call atoi).

Инструкция ret является последней в подпрограмме и ее исполнение приводит к возвращению в основную программу к инструкции, следующей за инструкцией

call, которая вызвала данную подпрограмму.

Я ввел в файл lab9-1.asm текст программы. Создал исполняемый файл и проверил его работу. (рис. ??).

```
kache1ukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ nasm -f elf lab9-1.asm
kache1ukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ ld -m elf_i386 -o lab9-1 lab9-1.o
kache1ukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ ./lab9-1
Введите x: 4
2x+7=15
kache1ukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
```

Далее я изменил 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 вычисляется выражение $f(g(x))$. Результат возвращается в основную программу для вывода результата на экран. (рис. ??).

```
kache1ukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ nasm -f elf lab9-1.asm
kache1ukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ ld -m elf_i386 -o lab9-1 lab9-1.o
kache1ukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ ./lab9-1
Введите x: 4
f(g(x))= 29
kache1ukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
```

4.2 Отладка программ с помощью GDB

Я создал файл lab9-2.asm с текстом программы из ТУИС.

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
```

Получил исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'. Загрузил исполняемый файл в отладчик gdb и проверил работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r) (рис. ??).

```

$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ ld -m elf_i386 -o lab9-2 lab9-2.o
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ gdb lab9-2
GNU gdb (Gentoo 15.2 vanilla) 15.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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 lab9-2...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/a/kachelukhaev1/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 11880) exited normally]
(gdb)

```

Для более подробного анализа программы установил брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустил её. (рис. ??).

```

(gdb) break _start
Breakpoint 1 at 0x08049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/a/kachelukhaev1/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
9      mov     eax, 4
(gdb)

```

Посмотрел дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. ??).

```

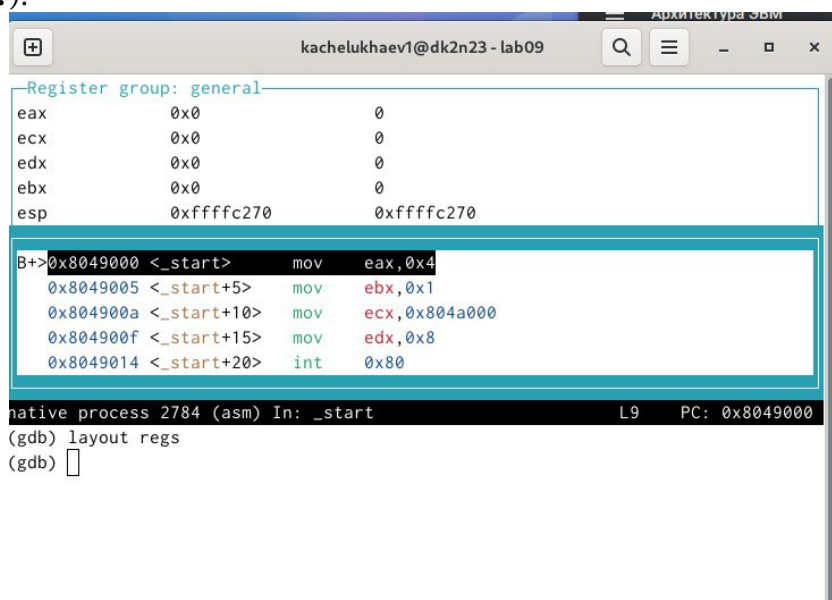
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     $0x4,%eax
      0x08049005 <+5>:      mov     $0x1,%ebx
      0x0804900a <+10>:     mov     $0x804a000,%ecx
      0x0804900f <+15>:     mov     $0x8,%edx
      0x08049014 <+20>:     int     $0x80
  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

```

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

```
(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) █
```

Включите режим псевдографики для более удобного анализа программы (рис. ??).



В этом режиме есть три окна: * В верхней части видны названия регистров и их текущие значения; * В средней части виден результат дисассимилирования программы; * Нижняя часть доступна для ввода команд.

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`).

Проверил это с помощью команды `info breakpoints` (кратко `i b`) (рис. ??).

```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab9-2.asm:9
        breakpoint already hit 1 time
(gdb) █
```

Установим еще одну точку останова по адресу инструкции. Посмотрел информацию о всех установленных точках останова (рис. ??).

```
native process 2784 (asm) In: _start      L9  PC: 0x8049000
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab9-2.asm:9
        breakpoint already hit 1 time
2        breakpoint      keep y   0x08049031 lab9-2.asm:20
(gdb) █
```

Посмотреть содержимое регистров также можно с помощью команды `info registers` (или `i r`). (рис. ??).

```
native process 2784 (asm) In: _start      L9  PC: 0x8049000
edi      0x0             0
eip      0x8049000       0x8049000 <_start>
eflags   0x202          [ IF ]
cs       0x23            35
ss       0x2b            43
ds       0x2b            43
es       0x2b            43
fs       0x0             0
--Type <RET> for more, q to quit, c to continue without paging--
█
```

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. (рис. ??).

```

native process 2784 (asm) In: _start          L9    PC: 0x8049000
ss          0x2b          43
ds          0x2b          43
es          0x2b          43
fs          0x0           0
--Type <RET> for more, q to quit, c to continue without paging--
gs          0x0           0
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb)

```

Также я посмотрел значение переменной msg2 по адресу (рис. ??).

```

native process 2784 (asm) In: _start          L9    PC: 0x8049000
es          0x2b          43
fs          0x0           0
--Type <RET> for more, q to quit, c to continue without paging--
gs          0x0           0
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)

```

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). Изменил первый символ переменной msg1 (рис. ??).

```

(gdb) set {char}&msg1= 'h'
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
No symbol "msg1" in current context.
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hhlllo, "
(gdb)

```

Далее я заменил символы во второй переменной msg2. (рис. ??).

```

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

```

Вывел в различных форматах значение регистра ebx. (рис. ??).


```
(gdb) p/s $ebx
$3 = 50
(gdb) p/t $ebx
$4 = 110010
(gdb) p/x $ebx
$5 = 0x32
(gdb) █
```

Завершил выполнение программы с помощью команды continue (сокращенно c) или stepi (сокращенно si) и вышел из GDB с помощью команды quit (сокращенно q).

Я скопировал файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки в файл с именем lab09-3.asm и создал исполняемый файл. (рис. ??).

```
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-
pc/lab09 $ nasm -f elf -g -l lab9-3.lst lab9-3.asm
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-
pc/lab09 $ ld -m elf_i386 -o lab9-3 lab9-3.o
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-
pc/lab09 $ █
```

Для загрузки в gdb программы с аргументами необходимо использовать ключ -args. Загрузил исполняемый файл в отладчик, указав аргументы. 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 останова перед первой инструкцией в программе и запустим ее. (рис. ??).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/a/kachelukhaev1/work/
study/2024-2025/Архитектура компьютера/arch-pc/lab09/lab9-3 4 5 Hell
o

Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) █
```

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

4.3 Задание для самостоятельной работы

1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. (рис. ??).

```
$ touch lab9-4.asm
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ nasm -f elf lab9-4.asm
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ ld -m elf_i386 lab9-4.o -o lab9-4
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ ./lab9-4 10
Сумма значений f(x): 60
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ █
```

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

```
SECTION .data
```

```
msg: db "Сумма значений f(x): ", 0
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
; ---- Извлечение количества аргументов и имени программы ----
```

```
pop ecx ; Извлекаем количество аргументов (включая имя программы)
```

```
pop edx ; Извлекаем имя программы (не используем)
```

```
sub ecx, 1 ; Уменьшаем количество аргументов (исключая имя программы)
```

```
mov esi, 0 ; Инициализируем сумму  $f(x)$  в esi
```

```
next_arg:
```

```
cmp ecx, 0
```

```
jz end_loop ; Если нет аргументов, перейти к выводу результата
```

```

; ---- Подготовка аргумента для подпрограммы ----
pop eax          ; Извлекаем текущий аргумент x
call atoi        ; Преобразовываем строку в число
push eax         ; Помещаем аргумент x в стек для передачи в подпрограмму
call calculate_f ; Вызываем подпрограмму calculate_f

add esp, 4       ; Очищаем стек после возврата из calculate_f (удаляем аргумент x)
add esi, eax     ; добавляем результат в сумму f(x)

loop next_arg

end_loop:
; ---- Вывод результата ----
mov eax, msg     ; Вывод сообщения "Сумма значений f(x): "
call sprint
mov eax, esi     ; Записываем сумму значений f(x) в регистр 'eax'
call iprintLF    ; Вывод результата
call quit        ; Выход из программы

; ---- Подпрограмма вычисления f(x) ----
calculate_f:
push ebp        ; Сохраняем ebp
mov ebp, esp    ; Устанавливаем ebp на вершину стека
mov eax, [ebp + 8]; Получаем x из стека (x является аргументом)

add eax, 10     ; Вычисляем 10+x
mov ebx, 3      ; Загружаем 3 для умножения
mul ebx         ; Умножаем (10 + x) на 3

```

```

mov esp, ebp      ; Восстанавливаем esp
pop ebp           ; Восстанавливаем ebp
ret               ; Возврат из подпрограммы

```

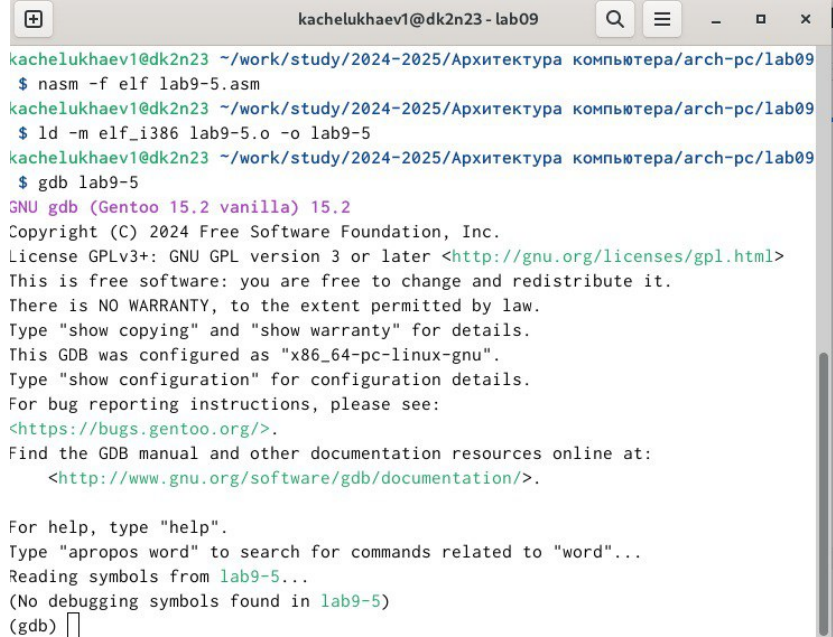
2. В листинге 9.3 приведена программа вычисления выражения $(3 + 2) \cdot 4 + 5$. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

```

#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

```

Я создал файл lab9-5.asm и начал отладку (рис. ??).



```
kachelukhaev1@dk2n23 - lab09
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ nasm -f elf lab9-5.asm
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ ld -m elf_i386 lab9-5.o -o lab9-5
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ gdb lab9-5
GNU gdb (Gentoo 15.2 vanilla) 15.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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 lab9-5...
(No debugging symbols found in lab9-5)
(gdb) □
```

Проверил его работу (рис. ??).

```
(gdb) q
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ nasm -f elf lab9-5.asm
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ ld -m elf_i386 lab9-5.o -o lab9-5
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ ./lab9-5
Результат: 25
kachelukhaev1@dk2n23 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
$ □
```

Исправленный код:

```
%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 ; ebx = 3 + 2 = 5
mov eax, ebx ; перемещаем результат сложения в eax
mov ecx, 4
mul ecx      ; eax = eax * ecx = 5 * 4 = 20
add eax, 5   ; eax = 20 + 5 = 25
mov edi, eax ; edi = 25

; ---- Вывод результата на экран
mov eax, div
call sprint
mov eax, edi
call iprintLF
call quit
```

5 Выводы

Я приобрел навык написания программ с использованием подпрограмм. И познакомился с методами отладки при помощи GDB и его основными возможностями.

Список литературы