

Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютеров и операционные системы

Лисенков Егор Романович

Содержание

1	Цель работы	1
2	Задание.....	1
3	Теоретическое введение	2
4	Выполнение лабораторной работы	3
4.1	Реализация подпрограмм в NASM.....	3
4.2	Отладка программ с помощью GDB.....	5
4.2.1	Добавление точек останова	7
4.2.2	Работа с данными программы в GDB	7
4.2.3	Обработка аргументов командной строки в GDB	11
4.3	Задания для самостоятельной работы	12
5	Выводы.....	15
6	Список литературы.....	15

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. Добавление точек останова.
4. Работа с данными программы в GDB.
5. Обработка аргументов командной строки в GDB.
6. Задания для самостоятельной работы.

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. 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.

GDB (GNU Debugger — отладчик проекта GNU) 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.

Команда `run` (сокращённо `r`) — запускает отлаживаемую программу в оболочке GDB.

Команда `kill` (сокращённо `k`) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. 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`).

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`.

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».

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

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой `disable`.

Обратно точка останова активируется командой `enable`.

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

Для продолжения остановленной программы используется команда `continue (c)`. Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число `N`, которое указывает отладчику проигнорировать `N – 1` точку останова (выполнение остановится на `N`-й точке).

Команда `stepi` (кратко `sl`) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию.

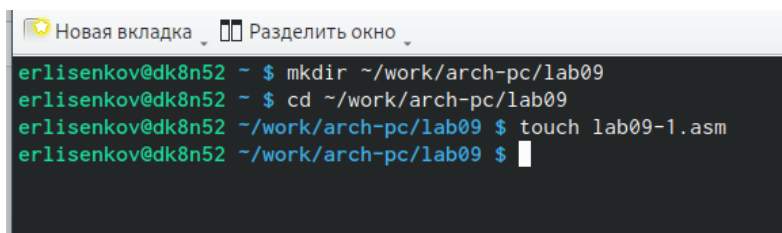
Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом. Если в программе встречается одинаковый участок кода, его можно оформить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

Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `esp` 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 `esp`. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией `call`.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

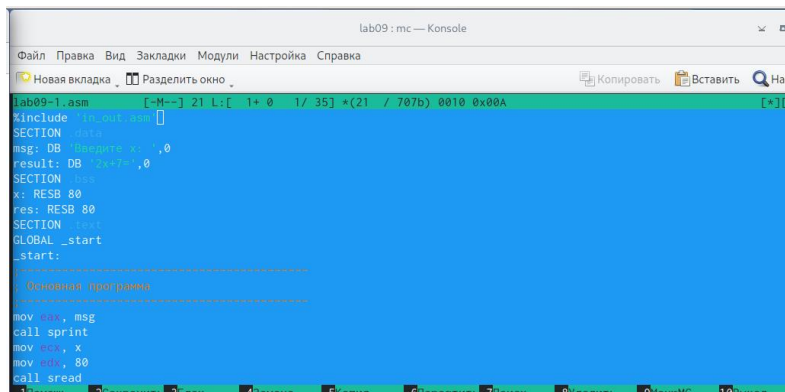
Создал каталог и перешёл в него + создаю файл `lab09-1.asm`. (рис. ??)



```
erlisenkov@dk8n52 ~ $ mkdir ~/work/arch-pc/lab09
erlisenkov@dk8n52 ~ $ cd ~/work/arch-pc/lab09
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ touch lab09-1.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $
```

Создание файлов для лабораторной работы

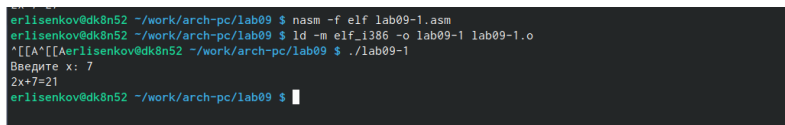
Записываю в файл `lab09-1.asm` текст программы с использованием листинга 9.1. (рис. ??)



```
lab09-1.asm [~M--] 21 L: [ 1+ 0 1/ 35] *(21 / 707b) 0010 0x00A [*]
#include "lab09-1.asm"
SECTION .text
msg: DB "Введите x: ", 0
result: DB "2x+7=", 0
SECTION .data
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
```

Ввод текста программы из листинга 9.1

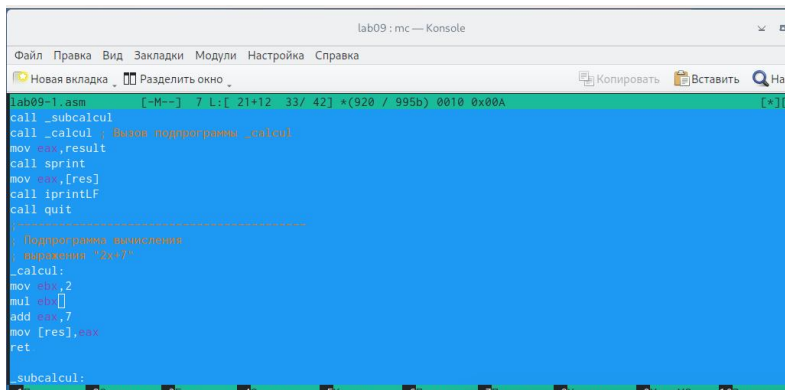
Проверяю его работу. (рис. ??)



```
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
^[[A^[[Aerlisenkov@dk8n52 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 7
2x+7=21
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $
```

Запуск исполняемого файла

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul` для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. (рис. ??)



```
lab09-1.asm [~M--] 7 L: [ 21+12 33/ 42] *(920 / 995b) 0010 0x00A [*]
call _subcalcul
call _calcul
mov eax, result
call _sprint
mov ebx, [res]
call _iprintLF
call _quit
; Подпрограмма вычисления
; выражения f(g(x))
_calcul:
mov ebx, 2
mul ebx, []
add ebx, 7
mov [res], ebx
ret
_subcalcul:
```

Изменение текста программы согласно заданию

Проверяю работу исполняемого файла. (рис. ??)

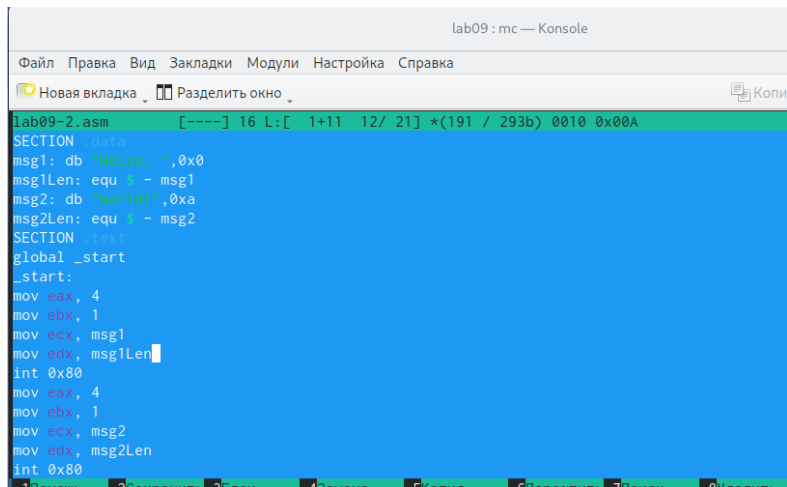


```
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 7
2x+7=47
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $
```

Запуск исполняемого файла

4.2 Отладка программ с помощью GDB

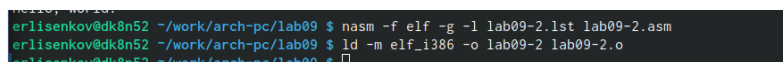
Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (рис. ??)



```
lab09-2.asm  [----] 16 L: [ 1+11 12/ 21] *(191 / 293b) 0010 0x00A
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
```

Ввод текста программы из листинга 9.2

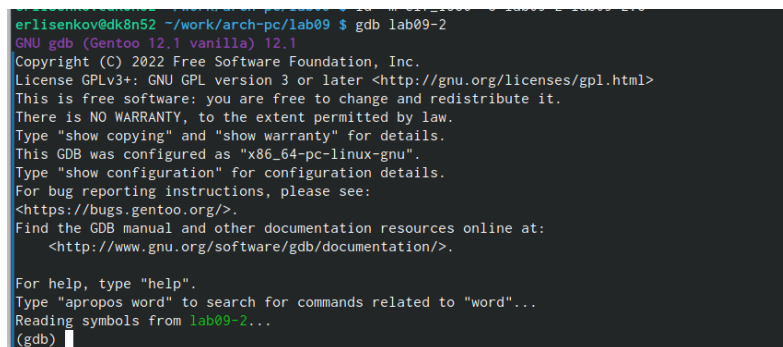
Получаю исполняемый файл для работы с GDB с ключом '-g'. (рис. ??)



```
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $
```

Получение исполняемого файла

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

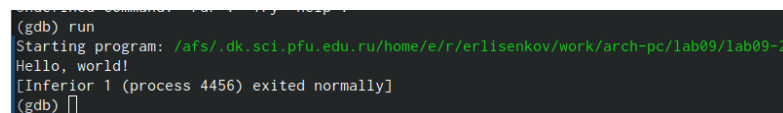


```
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 12.1 vanilla) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-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 lab09-2...
(gdb)
```

Загрузка исполняемого файла в отладчик

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



```
(gdb) run
Starting program: /afs/.dk.sc1.pfu.edu.ru/home/e/r/erlisenkov/work/arch-pc/lab09/lab09-2
Hello, world!
[[Inferior 1 (process 4456) exited normally]]
(gdb)
```

Проверка работы файла с помощью команды run

Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start` и запускаю её. (рис. ??)

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/r/erlisenkov/work/arch-pc/lab09/lab09-2

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

Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`, и переключаюсь на отображение команд с синтаксисом Intel, введя команду `set disassembly-flavor intel`. (рис. ??)

```
9      mov eax, 4
(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) 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) |
```

Использование команд `disassemble` и `disassembly-flavor intel`

В режиме АТТ имена регистров начинаются с символа `%`, а имена операндов с `$`, в то время как в Intel используется привычный нам синтаксис.

Включаю режим псевдографики для более удобного анализа программы с помощью команд `layout asm` и `layout regs`. (рис. ??)

```
lab09: gdb — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти

[ Register Values Unavailable ]

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

native process 4481 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb)
```

Включение режима псевдографики

4.2.1 Добавление точек останова

Проверяю, что точка останова по имени метки `_start` установлена с помощью команды `info breakpoints` и устанавливаю еще одну точку останова по адресу инструкции `mov ebx,0x0`. Просматриваю информацию о всех установленных точках останова. (рис. ??)

```
lab09: gdb — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти

[ Register Values Unavailable ]

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 edi,0x7

native process 4481 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(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)
```

Установка точек останова и просмотр информации о них

4.2.2 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды `stepi` и слежу за изменением значений регистров. (рис. ??)

```
lab09: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти

Register group: general
eax      0x4          4          ecx      0x0          0
edx      0x0          0          ebx      0x0          0
esp      0xffffc300   0xffffc300   ebp      0x0          0x0
esi      0x0          0          edi      0x0          0
eip      0x8049005     0x8049005   eflags   0x202         [ IF ]
cs       0x23         35          ss       0x2b         43
ds       0x2b         43          es       0x2b         43
fs       0x0          0          gs       0x0          0

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

native process 4481 In: _start L10 PC: 0x8049005
edi      0x0          0
eip      0x8049000     0x8049000 <_start>
eflags   0x202         [ IF ]
cs       0x23         35
ss       0x2b         43
--Type <RET> for more, q to quit, c to continue without paging-- 0x2b 43
es       0x2b         43
fs       0x0          0
gs       0x0          0
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) stepi
(gdb)
```

До использования команды *stepi*

(рис. ??)

```
lab09: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти

Register group: general
eax      0x4          4          ecx      0x804a000   134520832
edx      0x8          8          ebx      0x1          1
esp      0xffffc300   0xffffc300   ebp      0x0          0x0
esi      0x0          0          edi      0x0          0
eip      0x8049020     0x8049020 <_start+32>

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

native process 4481 In: _start L16 PC: 0x8049020
Unmatched single quote.
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)
```

После использования команды *stepi*

Изменились значения регистров *eax*, *ecx*, *edx* и *ebx*.

Просматриваю значение переменной *msg1* по имени с помощью команды *x/1sb &msg1* и значение переменной *msg2* по ее адресу. (рис. ??)


```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd160 0xffffd160
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>

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

native process 7430 In: _start L14 PC: 0x8049016
35
ss      0x2b      43
ds      0x2b      43
es      0x2b      43
fs      0x0      0
gs      0x0      0
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"

```

Просмотр значений переменных

С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2. (рис. ??)

```

(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}&msg2 = 'b'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "borld!\n\034"

```

Использование команды set

Вывожу в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра edx с помощью команды print p/F \$val. (рис. ??)

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd160 0xffffd160
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]

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

native process 7430 In: _start L14 PC: 0x8049016
(gdb) p/x $edx
$8 = 0x8
(gdb) p/t $edx
$9 = 1000
(gdb) p/c $edx
$10 = 8 '\b'
```

Вывод значения регистра в разных представлениях

С помощью команды set изменяю значение регистра ebx в соответствии с заданием. (рис. ??)

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd160 0xffffd160
ebp      0x0      0x0

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

native process 19345 In: _start L14 PC: 0x8049016
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb)
```

Использование команды set для изменения значения регистра

Разница вывода команд p/s \$ebx отличается тем, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется.

Завершаю выполнение программы с помощью команды continue и выхожу из GDB с помощью команды quit. (рис. ??)

```
Register group: general
eax      0x1      1
ecx      0x804a008 134520840
edx      0x7      7
ebx      0x1      1
esp      0xffffd160 0xffffd160
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049031 0x8049031 <_start+49>

0x8049020 <_start+32> mov    ecx,0x804a008
0x8049025 <_start+37> mov    edx,0x7
0x804902a <_start+42> int    0x80
0x804902c <_start+44> mov    eax,0x1
B> 0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80
0x8049038      add    BYTE PTR [eax],al
0x804903a      add    BYTE PTR [eax],al

native process 7430 In: _start      L20  PC: 0x8049031
$14 = 2
(gdb) c
Continuing.
world!
Breakpoint 2, _start () at lab09-2.asm:20
(gdb) q
A debugging session is active.

    Inferior 1 [process 7430] will be killed.

Quit anyway? (y or n)
```

Завершение работы GDB

4.2.3 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09-3.asm и создаю исполняемый файл. (рис. ??)

```
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Создание файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа `--args`. (рис. ??)

```
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Gentoo 12.1 vanilla) 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-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 lab09-3...
(gdb)
```

Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. ??)

```

Reading symbols from lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/r/erlisenkov/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргум
ент\ 3
Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)

```

Установка точки останова и запуск программы

Посматриваю вершину стека и позиции стека по их адресам. (рис. ??)

```

(gdb) x/x $esp
0xfffffc2b0: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xfffffc54f: "/afs/.dk.sci.pfu.edu.ru/home/e/r/erlisenkov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc596: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffc5a8: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffc5b9: "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffc5bb: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Просмотр значений, введенных в стек

Шаг изменения адреса равен 4, т.к количество аргументов командной строки равно 4.

4.3 Задания для самостоятельной работы

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

```

task1.asm [-----] 0 L: [ + 0 / 31 ] * (0 / 323b) 0037 0x025 [*]
%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
mov edi,5
call .next
.next:
pop eax
call atoi
add eax,2
mul edi

```

Написание кода подпрограммы

Запускаю код и проверяю, что она работает корректно. (рис. ??)

```

erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ nasm -f elf task1.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o task1 task1.o
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ ./task1 1 2 3
Результат: 45
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $

```

Запуск программы и проверка его вывода

Программа:

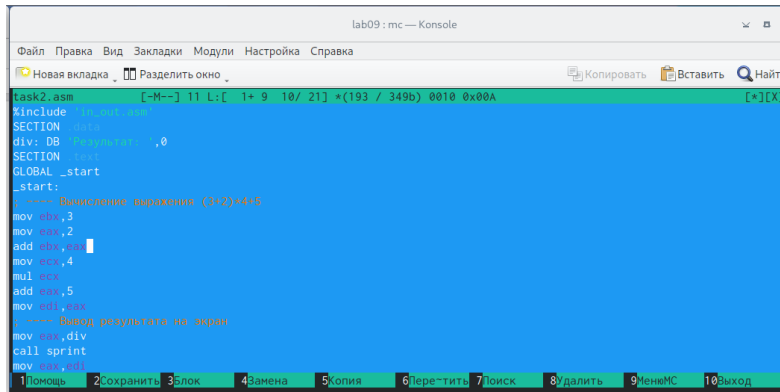
```

#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 mov edi,5 call .next
.next: pop eax call atoi add

```

```
eax,2 mul edi add esi,eax cmp ecx,0h jz .done loop .next .done: mov eax, msg call sprint mov
eax, esi call iprintLF call quit ret
```

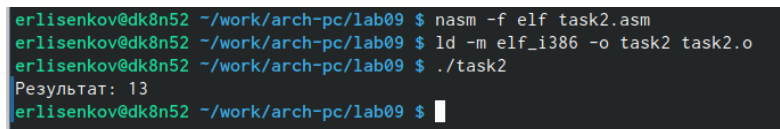
2. Ввожу в файл task1.asm текст программы из листинга 9.3. (рис. ??)



```
task2.asm [-M--] 11 L: [ 1+ 9 10/ 21] *(193 / 349b) 0010 0x00A [*][X]
#include "task1.asm"
SECTION .text
div: DB "div: ",0
SECTION .data
GLOBAL _start
_start:
; ----- Вычисление выражения: (3*2)*4*5
mov ebx,3
mov ecx,2
add ebx,ecx
mov ecx,4
mul ecx
add ecx,5
mov edi,eax
; ----- Вывод результата на экран
mov ebx,div
call sprint
mov eax,edi
```

Листинг 9.3

При корректной работе программы должно выводиться "25". Создаю исполняемый файл и запускаю его. (рис. ??)



```
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ nasm -f elf task2.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o task2 task2.o
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $ ./task2
Результат: 13
erlisenkov@dk8n52 ~/work/arch-pc/lab09 $
```

Запуск исполняемого файла

Видим, что в выводе мы получаем неправильный ответ.

Получаю исполняемый файл для работы с GDB, запускаю его и ставлю брейкпоинты для каждой инструкции, связанной с вычислениями. С помощью команды continue прохожусь по каждому брейкпоинту и слежу за изменениями значений регистров.

При выполнении инструкции mul ecx происходит умножение ecx на eax, то есть 4 на 2, вместо умножения 4 на 5 (регистр ebx). Происходит это из-за того, что стоящая перед mov ecx,4 инструкция add ebx,eax не связана с mul ecx, но связана инструкция mov eax,2. (рис. ??)

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd160 0xffffd160
ebp      0x0      0x0

B+ 0x80490f4 <_start+12> mov     ecx,0x4
B+ 0x80490f9 <_start+17> mul     ecx
B+ 0x80490fb <_start+19> add     ebx,0x5
b+ 0x80490fe <_start+22> mov     edi,ebx
0x8049100 <_start+24> mov     eax,0x804a000

native process 14573 In: _start L13 PC: 0x80490fb
Continuing.

Breakpoint 5, _start () at task2.asm:12
(gdb) c
Continuing.

Breakpoint 6, _start () at task2.asm:13
(gdb)

```

Причина ошибки

Мы нашли причину ошибки. (рис. ??)

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd160 0xffffd160
ebp      0x0      0x0

B+ 0x80490fb <_start+17> mul     ecx
B+ 0x80490fb <_start+19> add     ebx,5
B+ 0x80490fe <_start+22> mov     edi,ebx
0x8049100 <_start+24> mov     eax,0x804a000
0x8049105 <_start+29> call    0x804900f <sprint>

native process 14573 In: _start L14 PC: 0x80490fe
Continuing.

Breakpoint 6, _start () at task2.asm:13
(gdb) c
Continuing.

Breakpoint 7, _start () at task2.asm:14
(gdb)

```

Неверное изменение регистра

Исправляем ошибку, добавляя после add ebx,eax mov eax,ebx и заменяя ebx на eax в инструкциях add ebx,5 и mov edi,ebx. (рис. ??)

```

lab09: mc — Console
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
[+] Новая вкладка  [ ] Разделить окно  [ ] Копировать  [ ] Вставить  [ ] Найти
task2.asm  [-----] 11 L: [ 1+ 0 1/ 22] *(11 / 361b) 0110 0x00E  [*][X]
#include <stdio.h>
SECTION .text
div: DB 0x00000000,0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov ecx,2
add ebx,eax
mov ecx,ebx
mov ecx,4
mul ecx
add ecx,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
1 Помощь 2 Сохранить 3 Блок 4 Замена 5 Копия 6 Перезагрузить 7 Поиск 8 Удалить 9 Имена 10 Выход

```

Исправление ошибки

Также, вместо того, чтобы изменять значение eax, можно было изменять значение неиспользованного регистра edx.

Создам исп. файл и проверю его работу. Всё работает хорошо! (рис. ??)

```
erlisenkova@dk8n52 ~/work/arch-pc/lab09 $ nasm -f elf task2.asm
erlisenkova@dk8n52 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o task2 task2.o
erlisenkova@dk8n52 ~/work/arch-pc/lab09 $ ./task2
Результат: 25
erlisenkova@dk8n52 ~/work/arch-pc/lab09 $
```

Ошибка исправлена

Код программы:

```
%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
```

5 Выводы

Навыки, которые я получил, точно пригодятся мне в будущей работе!

6 Список литературы

Лабораторная работа №9. Понятие подпрограммы. Отладчик . Файл