

Отчёта по лабораторной работе № 9

Простейший вариант

Сахно Алёна Юрьевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	9.5. Задание для самостоятельной работы	23
	Список литературы	25

Список иллюстраций

4.1	Задали файл	9
4.2	Листинг 9.1. Пример программы с использованием вызова подпро- граммы	10
4.3	Результат с изменением	11
4.4	Листинг 9.2. Программа вывода сообщения Hello world!	12
4.5	Проверка работы	13
4.6	Установка брейпоинта	14
4.7	Проверка	16
4.8	Установка еще одной точки	17
4.9	Значение переменной msg1	18
4.10	Значение переменной msg2	19
4.11	Замени любой символ во второй переменной msg2.	20
4.12	Вывод а различных формах	20
4.13	Объяснение	22
5.1	Листинг 9.3. Программа вычисления выражения $(3 + 2) \cdot 4 + 5$. . .	24

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. Теоретическая часть
2. Выполнение лабораторной работы
3. Выполнения самостоятельной работы

3 Теоретическое введение

9.2.1. Понятие об отладке

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа:

- обнаружение ошибки;
- поиск её местонахождения;
- определение причины ошибки;
- исправление ошибки.

Можно выделить следующие типы ошибок:

- синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка;
- семантические ошибки — являются логическими и приводят к тому, что 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

1. Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и создайте файл lab09-1.asm:

```
mkdir ~/work/arch-pc/lab09
```

```
cd ~/work/arch-pc/lab09
```

```
touch lab09-1.asm
```

(рис.1 4.1).

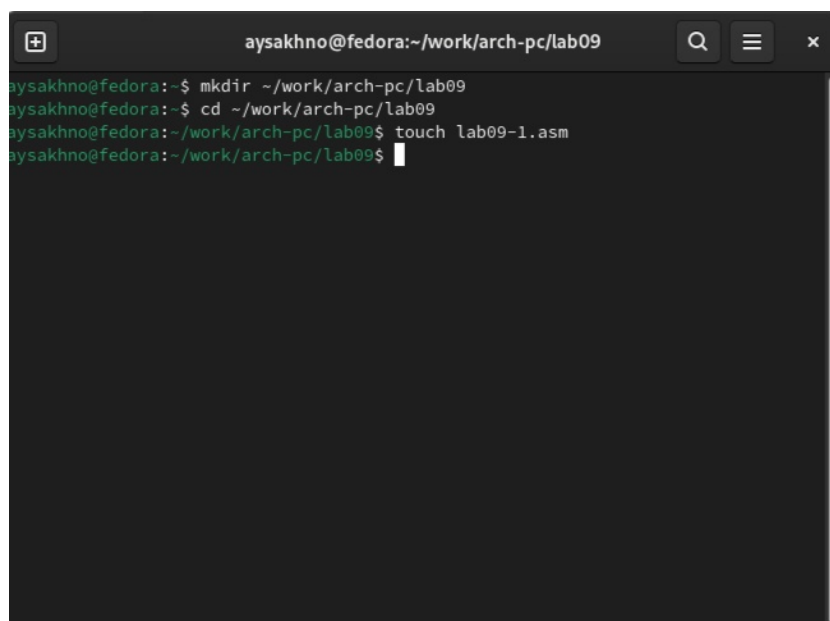
A screenshot of a terminal window with a dark background. The window title is 'aysakhno@fedora:~/work/arch-pc/lab09'. The terminal shows the following commands and their outputs: 'mkdir ~/work/arch-pc/lab09', 'cd ~/work/arch-pc/lab09', and 'touch lab09-1.asm'. The prompt is 'aysakhno@fedora:~/work/arch-pc/lab09\$'.

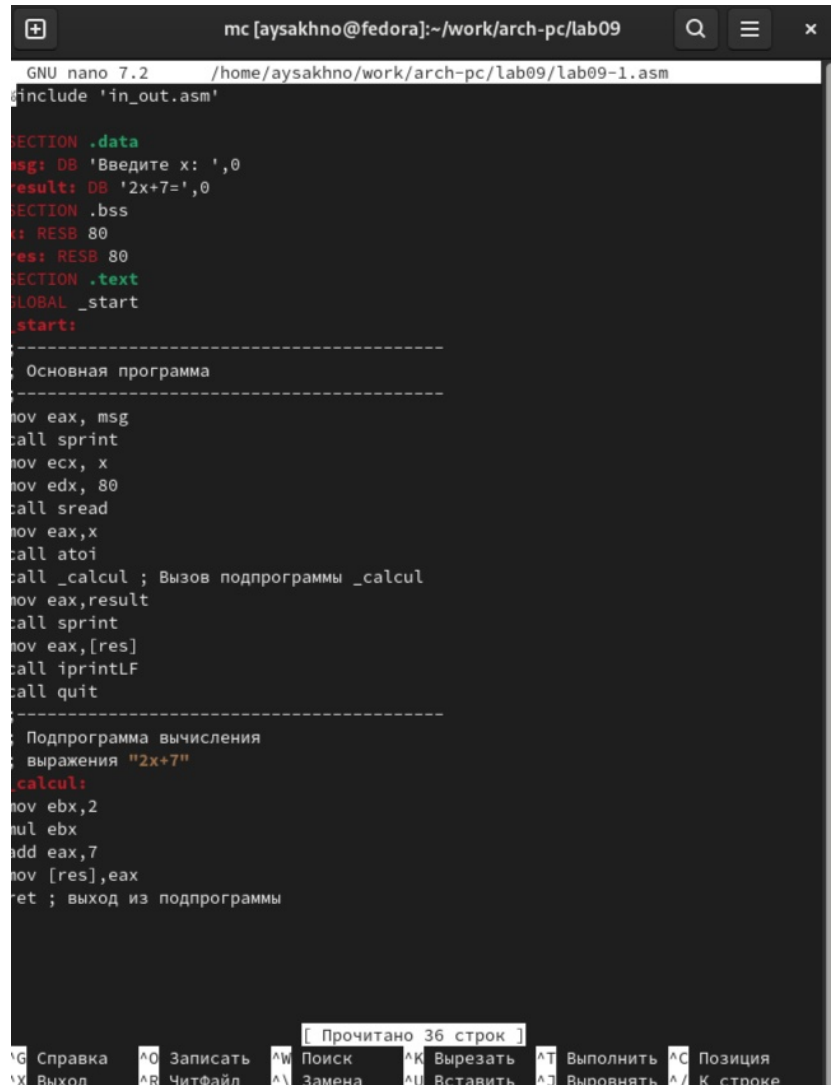
Рис. 4.1: Задали файл

2. В качестве примера рассмотрим программу вычисления арифметического выражения $\square(\square) = 2\square + 7$ с помощью подпрограммы _calcul. В данном примере

□ вводится с клавиатуры, а само выражение вычисляется в подпрограмме.

Внимательно изучите текст программы (Листинг 9.1

(рис.2 4.2).



```
GNU nano 7.2 /home/aysakhno/work/arch-pc/lab09/lab09-1.asm
#include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
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

[ Прочитано 36 строк ]
^G Справка ^O Записать ^M Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выводить ^_ К строке
```

Рис. 4.2: Листинг 9.1. Пример программы с использованием вызова 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). mov eax, msg ; вызов подпрограммы печати сообщения call sprint ; 'Введите x:' mov ecx, x mov edx, 80 call

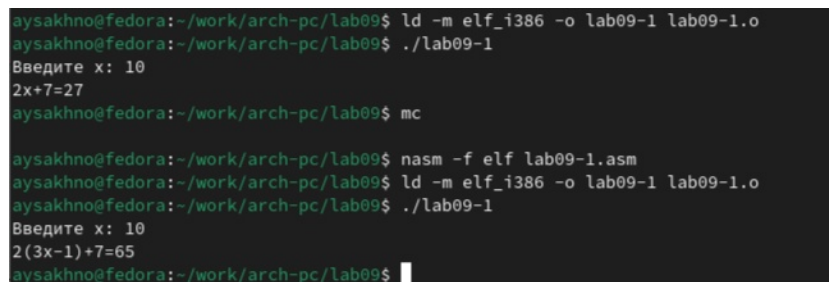
sread ; вызов подпрограммы ввода сообщения `mov eax,x` ; вызов подпрограммы преобразования `call atoi` ; ASCII кода в число, `eax=x`

После следующей инструкции `call _calcul`, которая передает управление подпрограмме `_calcul`, будут выполнены инструкции подпрограммы:

```
mov ebx,2 mul ebx add eax,7 mov [res],eax ret
```

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. Последние строки программы реализуют вывод сообщения (`call sprint`), результата вычисления (`call iprintLF`) и завершение программы (`call quit`). Введите в файл `lab09-1.asm` текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу. Измените текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $\square(\square(\square))$, где \square вводится с клавиатуры, $\square(\square) = 2\square + 7$, $\square(\square) = 3\square - 1$. Т.е. \square передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение $\square(\square)$, результат возвращается в `_calcul` и вычисляется выражение $\square(\square(\square))$. Результат возвращается в основную программу для вывода результата на экран

(рис.3 4.3).



```
aysakhno@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
aysakhno@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 10
2x+7=27
aysakhno@fedora:~/work/arch-pc/lab09$ mc

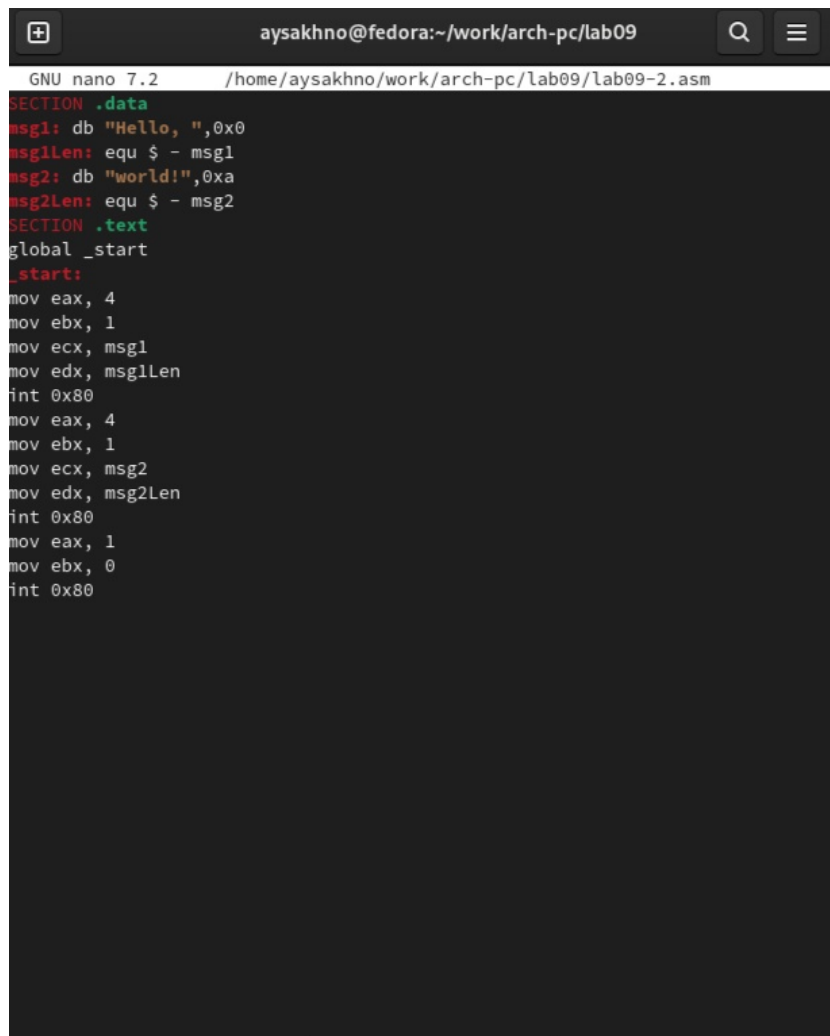
aysakhno@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
aysakhno@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
aysakhno@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 10
2(3x-1)+7=65
aysakhno@fedora:~/work/arch-pc/lab09$
```

Рис. 4.3: Результат с изменением

9.4.2. Отладка программ с помощью GDB

Создайте файл `lab09-2.asm` с текстом программы из Листинга 9.2. (Программа печати сообщения `Hello world!`):

(рис.4 4.4).

The image shows a terminal window with a dark background. At the top, the window title is 'aysakhno@fedora:~/work/arch-pc/lab09'. Below the title bar, the text 'GNU nano 7.2 /home/aysakhno/work/arch-pc/lab09/lab09-2.asm' is visible. The main content of the terminal is assembly code. It starts with a section named '.data' containing two strings: 'msg1' with the value 'Hello, ' and 'msg2' with the value 'world!'. Each string is followed by its length. Then, a section named '.text' is shown, starting with a global symbol '_start'. The code then defines '_start' as a procedure that prints 'Hello, ' and 'world!' using the 'int 0x80' instruction. The code is as follows:

```
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. 4.4: Листинг 9.2. Программа вывода сообщения Hello world!

Получите исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'.

```
nasm -f elf -g -l lab09-2.lst lab09-2.asm
```

```
ld -m elf_i386 -o lab09-2 lab09-2.o
```

Загрузите исполняемый файл в отладчик gdb:

user@dk4n31:~\$ gdb lab09-2 Проверьте работу программы, запустив ее в оболочке GDB с помощью команды run (со- кращённо r):

```
(gdb) run Starting program: ~/work/arch-pc/lab09/lab09-2 Hello, world! [Inferior 1
```

(process 10220) exited normally] (gdb)

(рис.5 4.5).

```
aysakhno@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 15.1-1.fc40
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb) run
Starting program: /home/aysakhno/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 5623) exited normally]
(gdb)
```

Рис. 4.5: Проверка работы

Для более подробного анализа программы установите брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустите её. (gdb) `break _start`

Breakpoint 1 at 0x8049000: file lab09-2.asm, line 12. (gdb) `run` Starting program:
~/work/arch-pc/lab09/lab09-2 Breakpoint 1, `_start ()` at lab09-2.asm:12 12 mov eax, 4
(рис.6 4.6).

```
(guv) q
aysakhno@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 15.1-1.fc40
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb) run
Starting program: /home/aysakhno/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 5623) exited normally]
(gdb) break _start
Breakpoint 1 at 0x08049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/aysakhno/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
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
```

Рис. 4.6: Установка брейпoinта

Посмотрите дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (`(gdb) disassemble _start`)

Переключитесь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (`(gdb) set disassembly-flavor intel`) (`(gdb) disassemble _start`)

Перечислите различия отображения синтаксиса машинных команд в режимах АТТ и Intel. Включите режим псевдографики для более удобного анализа программы (рис. 9.2): (`(gdb) layout asm`) (`(gdb) layout regs`) 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.

9.4.2.1. Добавление точек останова 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`). Проверьте это с помощью команды `info breakpoints` (кратко `i b`):
(`gdb`) `info breakpoints`

(рис. 7 4.7).

```

(gdb) run
Starting program: /home/aysakhno/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
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     $0x80
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:    mov     eax,0x4
      0x08049005 <+5>:    mov     ebx,0x1
      0x0804900a <+10>:   mov     ecx,0x804a000
      0x0804900f <+15>:   mov     edx,0x8
      0x08049014 <+20>:   int     0x80
      0x08049016 <+22>:   mov     eax,0x4
      0x0804901b <+27>:   mov     ebx,0x1
      0x08049020 <+32>:   mov     ecx,0x804a008
      0x08049025 <+37>:   mov     edx,0x7
      0x0804902a <+42>:   int     0x80
      0x0804902c <+44>:   mov     eax,0x1
      0x08049031 <+49>:   mov     ebx,0x0
      0x08049036 <+54>:   int     0x80
End of assembler dump.
(gdb) layout asm
aysakhno@fedora: ~/work/arch-pc/lab09$

```

Рис. 4.7: Проверка

Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции (см. рис. 9.3). Определите адрес предпоследней инструкции (`mov ebx,0x0`) и установите точку останова. `(gdb) break *` Посмотрите информацию о всех установленных точках останова: `(gdb) i b` (рис.8 4.7).


```
aysakhno@fedora:~/work/arch-pc/lab09
--Register group: general--
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd070 0xffffd070
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35

0x80490b6 add BYTE PTR [eax],al
0x80490b8 add BYTE PTR [eax],al
0x80490ba add BYTE PTR [eax],al
0x80490bc add BYTE PTR [eax],al
0x80490be add BYTE PTR [eax],al
0x80490c0 add BYTE PTR [eax],al
0x80490c2 add BYTE PTR [eax],al
0x80490c4 add BYTE PTR [eax],al
0x80490c6 add BYTE PTR [eax],al
0x80490c8 add BYTE PTR [eax],al
0x80490ca add BYTE PTR [eax],al
0x80490cc add BYTE PTR [eax],al

native process 5656 (asm) In: _start L9 PC: 0x8049000
(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) info breakpoints
num   Type             Disp Enb Address      What
-----
1     breakpoint         keep y 0x08049000 lab09-2.asm:9
      breakpoint already hit 1 time
(gdb) 
```

Рис. 4.8: Установка еще одной точки

9.4.2.2. Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных.

Выполните 5 инструкций с помощью команды `stepi` (или `si`) и проследите за изменением значений регистров. Значения каких регистров изменяются? Посмотреть содержимое регистров также можно с помощью команды `info registers` (или `i r`).

```
(gdb) info registers
```

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 пере- менной. Посмотрите значение переменной msg1 по имени (gdb) x/1sb &msg1 0x804a000 : “Hello,”

(рис.9 4.9).

The screenshot shows a GDB terminal window with the following content:

```

aysakhno@fedora:~/work/arch-pc/lab09
--Register group: general--
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd070 0xffffd070
ebp      0x0      0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202     [ IF ]
cs       0x23     35

0x80490b6 add BYTE PTR [eax],al
0x80490b8 add BYTE PTR [eax],al
0x80490ba add BYTE PTR [eax],al
0x80490bc add BYTE PTR [eax],al
0x80490be add BYTE PTR [eax],al
0x80490c0 add BYTE PTR [eax],al
0x80490c2 add BYTE PTR [eax],al
0x80490c4 add BYTE PTR [eax],al
0x80490c6 add BYTE PTR [eax],al
0x80490c8 add BYTE PTR [eax],al
0x80490ca add BYTE PTR [eax],al
0x80490cc add BYTE PTR [eax],al

native process 5656 (asm) In: _start L9 PC: 0x8049000
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--
fs       0x2b     43
gs       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb)
  
```

Рис. 4.9: Значение переменной msg1

Посмотрите значение переменной msg2 по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрите инструкцию mov esx,msg2 которая записывает в регистр esx адрес переменной msg2 (рис. 9.4) Архитектура ЭВМ (рис.10 4.10).

```

aysakhno@fedora:~/work/arch-pc/lab09
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd070 0xffffd070
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35

0x80490b6 add BYTE PTR [eax],al
0x80490b8 add BYTE PTR [eax],al
0x80490ba add BYTE PTR [eax],al
0x80490bc add BYTE PTR [eax],al
0x80490be add BYTE PTR [eax],al
0x80490c0 add BYTE PTR [eax],al
0x80490c2 add BYTE PTR [eax],al
0x80490c4 add BYTE PTR [eax],al
0x80490c6 add BYTE PTR [eax],al
0x80490c8 add BYTE PTR [eax],al
0x80490ca add BYTE PTR [eax],al
0x80490cc add BYTE PTR [eax],al

native process 5656 (asm) In: _start L9 PC: 0x8049000
eflags 0x202 [ IF ]
cs      0x23 35
ss      0x2b 43
--Type <RET> for more, q to quit, c to continue without paging--
ds      0x2b 43
es      0x2b 43
fs      0x0 0
gs      0x0 0
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb &msg2
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 4.10: Значение переменной msg2

Посмотрите значение переменной msg2 по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрите инструкцию mov ecx,msg2 которая записывает в регистр ecx адрес переменной msg2 (рис. 9.4). Рис. 9.4. Отображение содержимого памяти 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). Измените первый символ переменной msg1 (рис. 9.5): (gdb) set {char}msg1='h' (gdb) x/1sb &msg1 0x804a000 : "hello," (gdb)

Замените любой символ во второй переменной msg2. Чтобы посмотреть значения регистров используется команда print /F (перед именем регистра обязательно ставится префикс \$) (рис. 9.6): p/F \$

(рис.11 4.11).

```
(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hhlllo, "
```

Рис. 4.11: Замена любого символа во второй переменной msg2.

Выведите в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. С помощью команды set измените значение регистра ebx: (gdb) set \$ebx='2' (gdb) p/s \$ebx

(рис.12 4.12).

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

Рис. 4.12: Вывод в различных форматах

\$3 = 50 (gdb) set \$ebx=2 (gdb) p/s \$ebx \$4 = 2 (gdb) Объясните разницу вывода команд p/s \$ebx. Завершите выполнение программы с помощью команды continue (сокращенно c) или stepi (сокращенно si) и выйдите из GDB с помощью команды quit (сокращенно q).

9.4.2.3. Обработка аргументов командной строки в GDB

Скопируйте 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:

```
cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
```

Создайте исполняемый файл.

```
nasm -f elf -g -l lab09-3.lst lab09-3.asm ld -m elf_i386 -o lab09-3 lab09-3.o
```

Для загрузки в gdb программы с аргументами необходимо использовать ключ -args. Загрузите исполняемый файл в отладчик, указав аргументы:

```
gdb -args lab09-3 аргумент1 аргумент 2 'аргумент 3'
```

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.

```
(gdb) b _start (gdb) run
```

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

```
(gdb) x/x $esp 0xffffd200: 0x05
```

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и 'аргумент 3'.

Посмотрите 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.

```
(gdb) x/s *(void**)(esp + 4)0xffffd358 : " /lab09 - 3"(gdb)x/s * (void * *) (esp + 8)
0xffffd3bc: "аргумент1" (gdb) x/s *(void**)(esp + 12)0xffffd3ce : ""(gdb)x/s * (void *
*) (esp + 16) 0xffffd3df: "2" (gdb) x/s *(void**)(esp + 20)0xffffd3e1 : "3"(gdb)x/s *
(void * *) (esp + 24) 0x0: <error: Cannot access memory at address 0x0> (gdb) Объ-
ясните, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.)
(рис.13 4.13).
```

```

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 7.
(gdb) run
Starting program: /home/aysakhno/work/arch-pc/lab09/lab09-3 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 lab09-3.asm:7
7      pop есх ; Извлекаем из стека в `есх` количество
(gdb) x/x $esp
0xffffd030:  0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd1f4:  "/home/aysakhno/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd21e:  "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd230:  "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd241:  "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd243:  "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:  <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 4.13: Объяснение

5 9.5. Задание для самостоятельной работы

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

(рис.14 5.1).

```

GNU nano 7.2 /home/aysakhno/work/arch-pc/lab09/lab8
#include 'in_out.asm'

SECTION .data
div: DB 'Результат: ', 0

SECTION .text
GLOBAL _start
_start:

    mov ebx, 3
    mov eax, 2
    add ebx, eax
    mov eax, ebx
    mov ecx, 4
    mul ecx
    add eax, 5
    mov edi, eax

    mov eax, div
    call sprint
    mov eax, edi
    call iprintLF

    call quit

```

Рис. 5.1: Листинг 9.3. Программа вычисления выражения $(3 + 2) \cdot 4 + 5$

(рис.15 ??).

```

aysakhno@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab8-1.lst lab8-1.asm
aysakhno@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab8-1 lab8-1.o
aysakhno@fedora:~/work/arch-pc/lab09$ ./lab8-1
Результат: 25

```

Выводы

Я приобрела навык написания программ с использованием подпрограмм. По-знакомилась с методами отладки при помощи GDB и его основными возможностями

Список литературы

:::{#refs}::: https://esystem.rudn.ru/pluginfile.php/2089096/mod_resource/content/0/%D0%9B%D