РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 9

дисциплина: Архитектура компьютера

Студент: Семенов Богдан

Группа: НКАбд-05-25

МОСКВА

2025 г.

Содержание

Список иллюстраций

1 Цель работы

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

2 Задание

~ Освоение работы с подпрограммами в NASM

~ Освоение отладчика GDB

~ Написать программу вычисления арифметического выражения с использованием подпрограмм

~ Написать программу с вложенными подпрограммами

~ Отладить программу вывода сообщения с помощью GDB

~ Исследовать расположение аргументов командной строки в стеке

~ Исправить ошибку в программе вычисления выражения с помощью отладчика

~ Преобразовать ранее написанную программу с использованием подпрограмм

~ Найти и исправить ошибку в программе с помощью GDB

3 Теоретическое введение

Подпрограмма — это изолированный фрагмент кода, предназначенный для решения конкретной задачи. Он активируется по вызову (call) и возвращает управление (ret). Этот подход делает код многократно используемым, более чистым и удобным для понимания.

Отладка — процесс поиска и исправления ошибок. GDB (GNU Debugger) — инструмент для:

~ Пошагового выполнения программ

~ Установки точек останова

~ Просмотра и изменения регистров/памяти

~ Анализа стека

Ключевые команды GDB: break, run, stepi, info registers, x, set. Для эффективной отладки программа компилируется с ключом -g.

4 Выполнение лабораторной работы

1. Создадим каталог для выполнения лабораторной работы №9, перейдем в него и создадим файл lab9-1.asm

Рис 1

2. Введем в файл lab9-1.asm текст программы из листинга 9.1. Создадим исполняемый файл и проверим его работу.

Открыть

lab9-
~/work/arc

```
%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:
    mov eax, msg
    call sprint
    mov ecx, x
    mov edx, 80
    call sread
    mov eax, x
    call atoi
    call _calcul
    mov eax, result
    call sprint
    mov eax, [res]
    call iprintLF
    call quit

_calcul:
    mov ebx, 2
    mul ebx
```

Рис 2

```
bsemenov@fedora:~/work/arch-pc/lab09$
bsemenov@fedora:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
bsemenov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
bsemenov@fedora:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 7
2x+7=21
bsemenov@fedora:~/work/arch-pc/lab09$
```

Рис 3

3. Изменим текст программы, добавив под программу _ subcalcul в под программу _calcul, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$.

```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(g(x)) = ',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
    mov eax, msg
    call sprint
    mov ecx, x
    mov edx, 80
    call sread
    mov eax, x
    call atoi
    call _calcul
    mov eax, result
    call sprint
    mov eax, [res]
    call iprintLF
    call quit
_calcul:
    push eax
    call _subcalcul
    mov ebx, 2
    mul ebx
    add eax, 7

    mov [res], eax
    pop eax
    ret
_subcalcul:
    mov ebx, [esp+4]
    mov eax, ebx
    mov ecx, 3
    mul ecx
    sub eax, 1
    ret
```

Рис 4

```
bsemenov@fedora:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
bsemenov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
bsemenov@fedora:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 7
2(3x-1)+7=47
bsemenov@fedora:~/work/arch-pc/lab09$
```

Рис 5

4. Создадим файл lab9-2.asm с текстом программы из Листинга 9.2. (Программа написания сообщения Hello world!)

```
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 6

5. Загружаем исполняемый файл в отладчик gdb

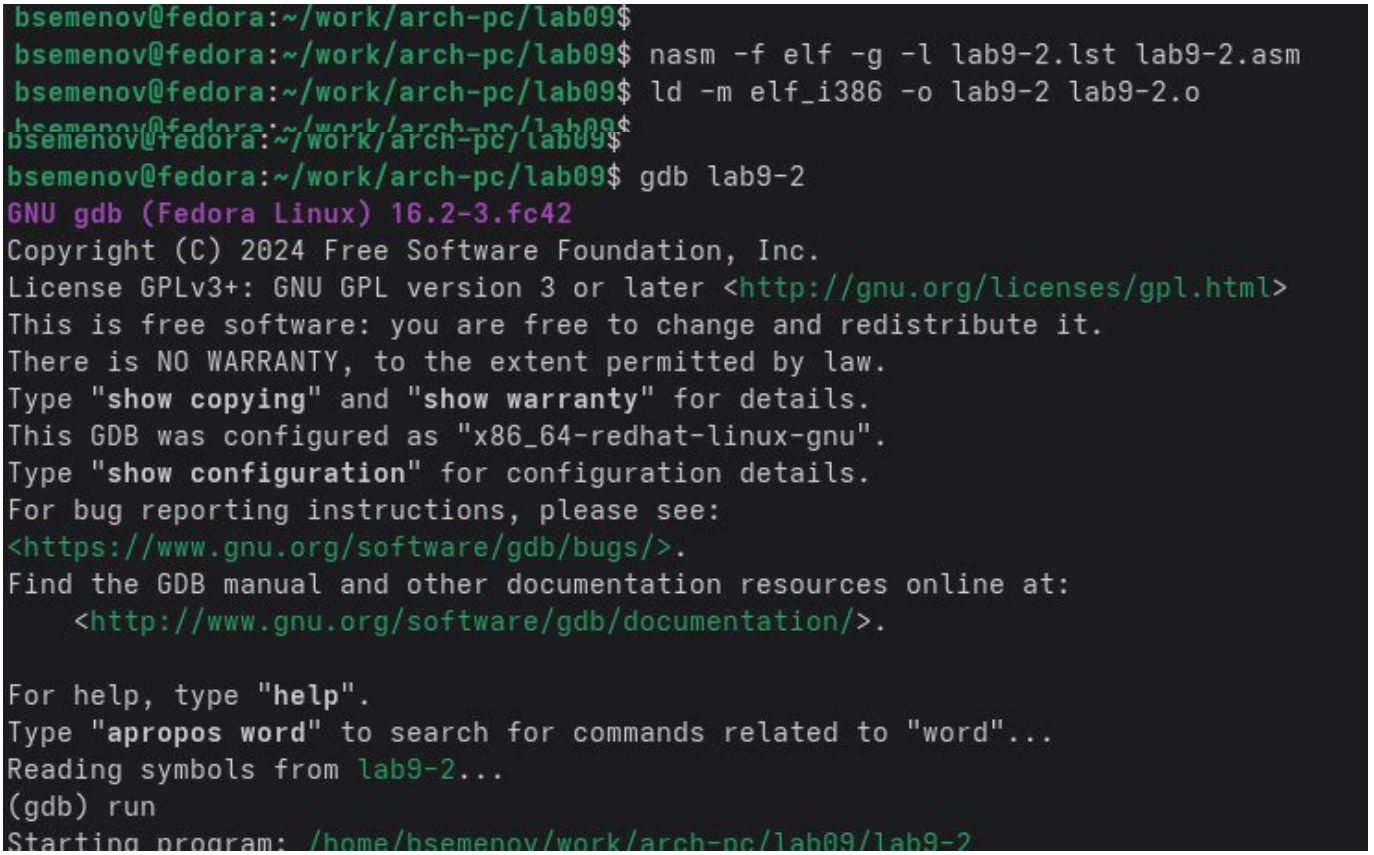
bsemenov@fedora:~/work/arch-pc/lab09$
bsemenov@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
bsemenov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
bsemenov@fedora:~/work/arch-pc/lab09$
bsemenov@fedora:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 16.2-3.fc42
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 lab9-2...
(gdb) run
Starting program: /home/bsemenov/work/arch-pc/lab09/lab9-2

Рис 7

6. Проверим работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r)

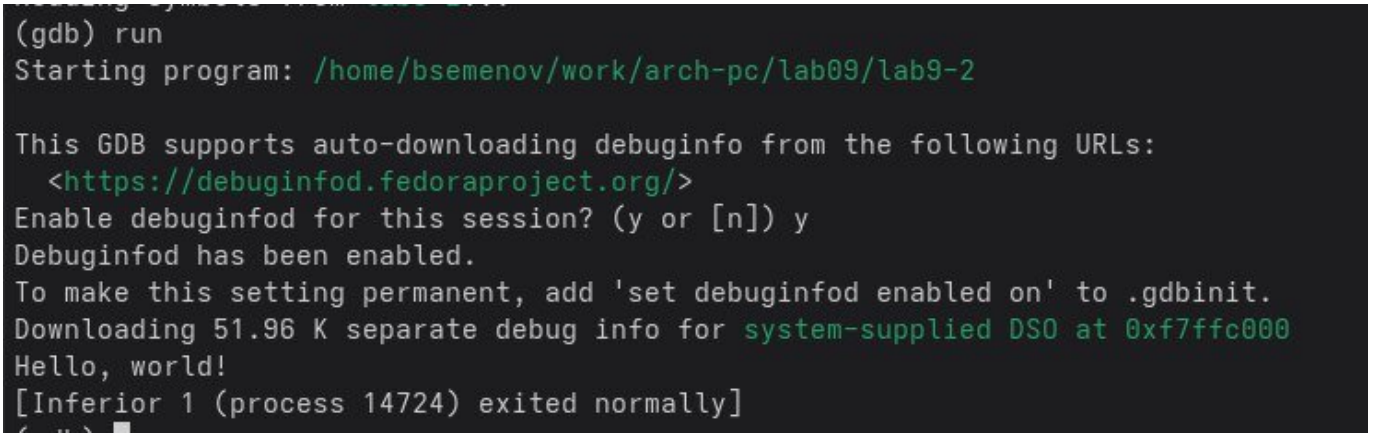

Рис 8

7. Для более подробного анализа программы установим брейк поинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустим её

Рис 9

8. Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки _start

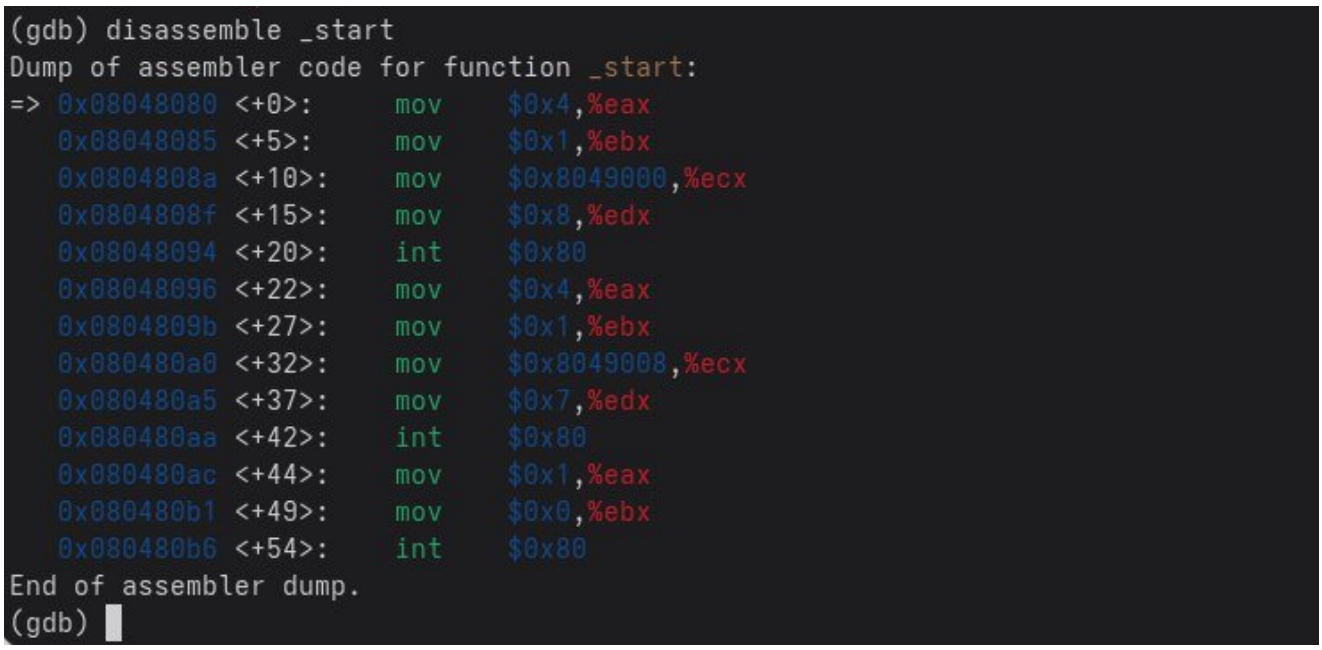

Рис 10

9. Переключимся на отображение команд с Intel’овским синтаксисом,введя команду set disassembly-flavor intel

Рис 11

10. Перечислим различия синтаксиса машинных команд в режимах ATT и Intel. Для наглядности будем использовать режим псевдографики, чтобы удобнее анализировать программу.

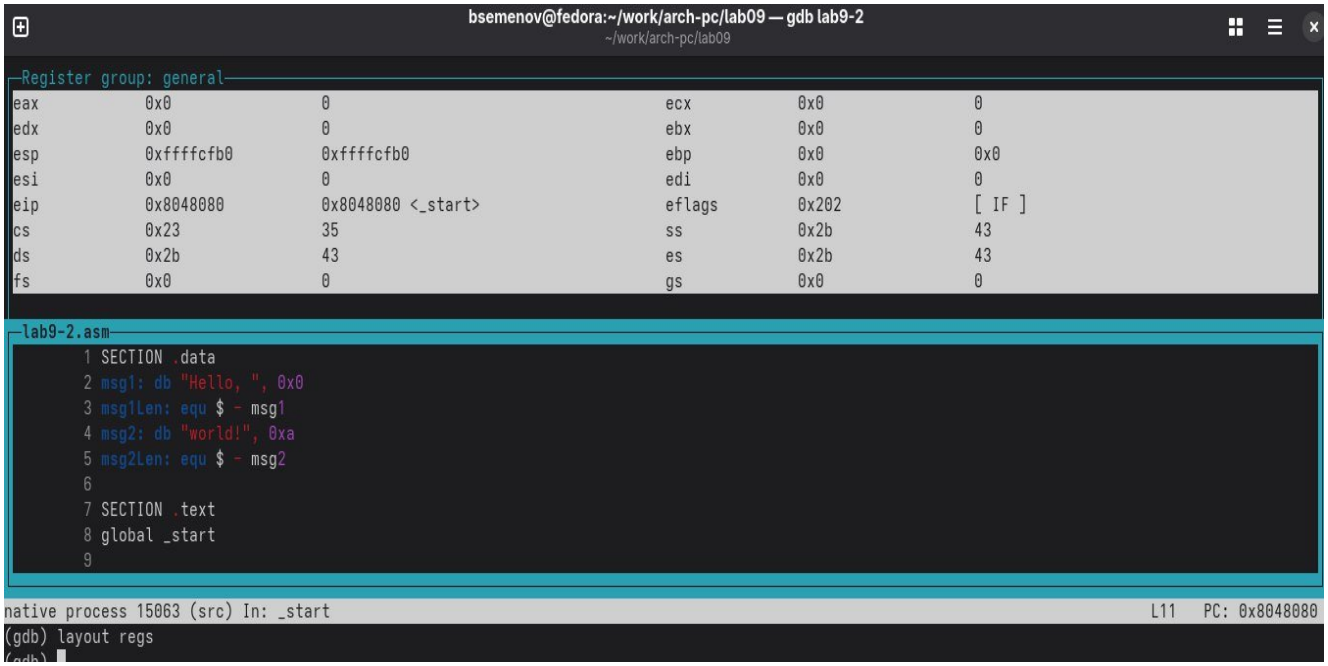

Рис 12

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

Проверим это с помощью команды info breakpoints (кратко i b)

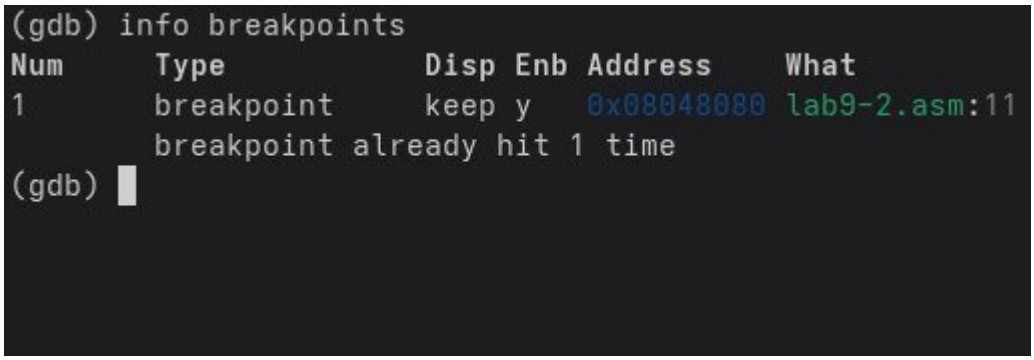

Рис 13

12. Поставим ещё одну точку останова по адресу инструкции. Адрес этой инструкции отображается в средней части экрана, в левом столбце соответствующей строки.

Определим адрес предпоследней инструкции (mov ebx, 0x0) и установим точку останова

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08048080 <+0>:     mov    $0x4,%eax
   0x08048085 <+5>:     mov    $0x1,%ebx
   0x0804808a <+10>:    mov    $0x8049000,%ecx
   0x0804808f <+15>:    mov    $0x8,%edx
   0x08048094 <+20>:    int    $0x80
   0x08048096 <+22>:    mov    $0x4,%eax
   0x0804809b <+27>:    mov    $0x1,%ebx
   0x080480a0 <+32>:    mov    $0x8049008,%ecx
   0x080480a5 <+37>:    mov    $0x7,%edx
   0x080480aa <+42>:    int    $0x80
   0x080480ac <+44>:    mov    $0x1,%eax
   0x080480b1 <+49>:    mov    $0x0,%ebx
   0x080480b6 <+54>:    int    $0x80
End of assembler dump.
(gdb)
(gdb) break *0x080480b1
Breakpoint 2 at 0x80480b1: file lab9-2.asm, line 24.
(gdb)
```

Рис 14

13. Посмотрим информацию о всех установленных точках останова

```
(gdb) i b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08048080 lab9-2.asm:11
        breakpoint already hit 1 time
2       breakpoint     keep y   0x080480b1 lab9-2.asm:24
(gdb)
```

Рис 15

14. Отладчик позволяет просматривать содержимое ячеек памяти и регистров, а также вручную изменять значения регистров и переменных при необходимости. Выполним 5 инструкций, используя команду ‘stepi’ (или ‘si’), и отследим изменения значений в регистрах.

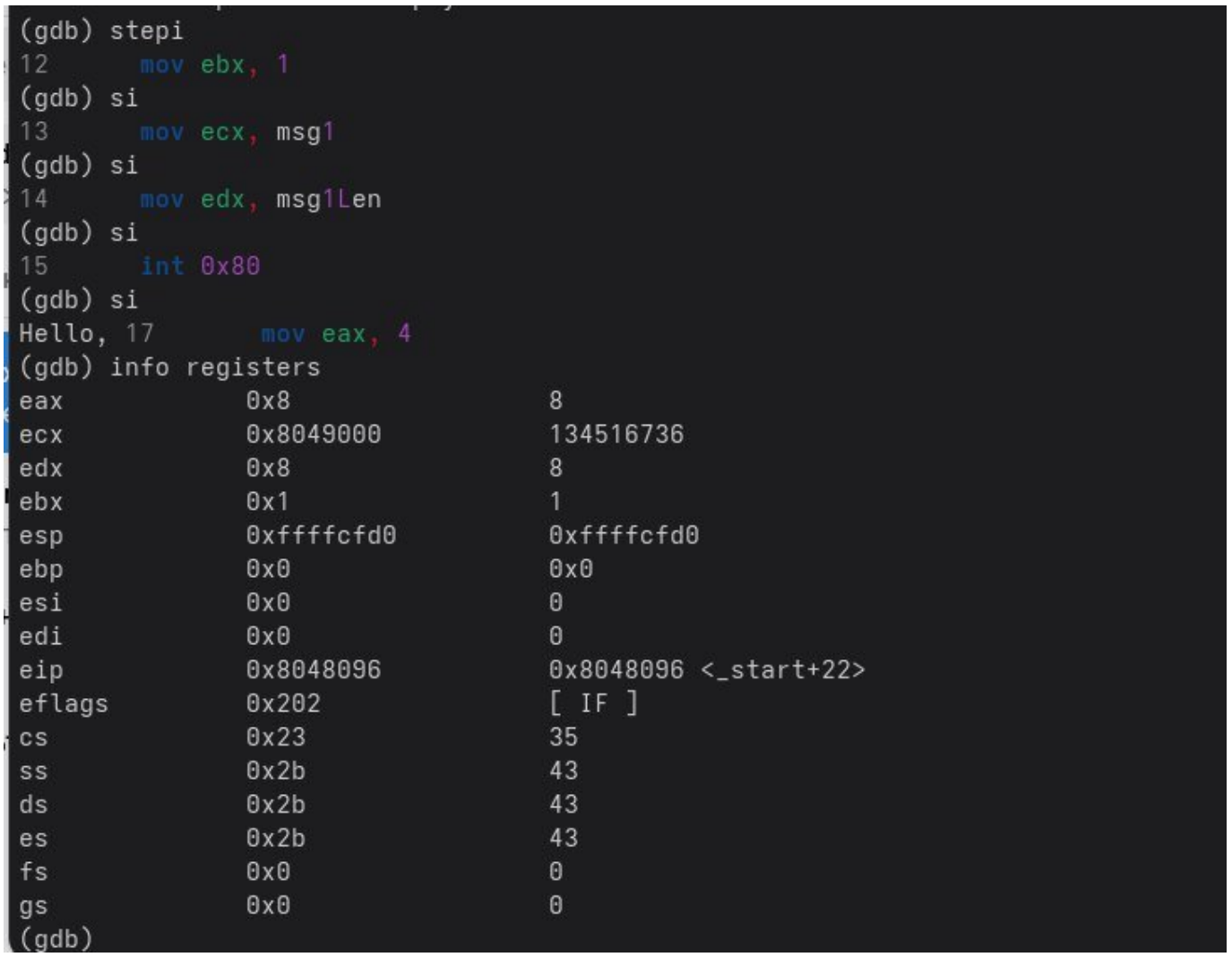

Рис 16

15. Посмотрите значение переменной msg1 по имени

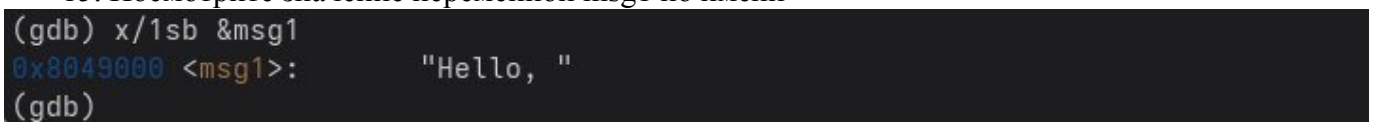

Рис 17

16. Посмотрим значение переменной ‘msg2’ по её адресу. Адрес переменной можно определить из дизассемблированной инструкции. Для этого обратимся к команде ‘mov ecx, msg2’, которая загружает в регистр ‘ecx’ адрес переменной ‘msg2’.

```
(gdb) x/1sb &msg1
0x8049000 <msg1>:       "Hello, "
(gdb) x/1sb 0x8049008
0x8049008 <msg2>:       "world!\n\034"
(gdb)
```

Рис 18

17. Изменим первый символ переменной msg1

```
0x8049008 <msg2>:       "world!\n\034"
(gdb) set {char}0x8049000 = 'h'
(gdb) x/1sb &msg1
0x8049000 <msg1>:       "hello, "
(gdb)
```

Рис 19

18. Заменим любой символ во второй переменной msg2.

```
0x8049008 <msg2>:       "world!\n\034"
(gdb) set {char}0x8049008 = 'W'
(gdb) x/1sb 0x8049008
0x8049008 <msg2>:       "World!\n\034"
(gdb)
```

Рис 20

Выведем значение регистра ‘edx’ в различных форматах: в шестнадцатеричном, двоичном и символьном представлении.

Рис 21

19. Объясним разницу вывода команд p/s $ebx.

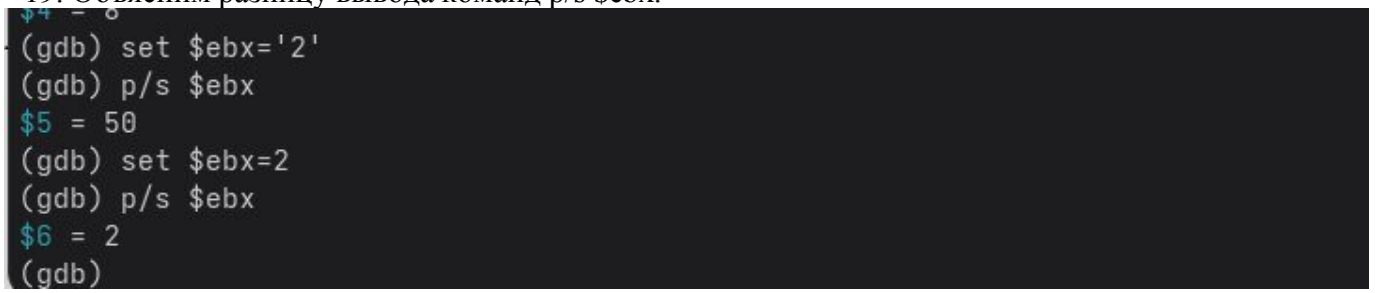

Рис 22

'2' - это символ (ASCII код 50),

а 2 - это число.

20. Завершим выполнение программы командой ‘continue’ (или ‘c’). Затем выйдем из отладчика GDB с помощью команды ‘quit’.

```
(gdb) continue
Continuing.
World!

Breakpoint 2, _start () at lab9-2.asm:24
24      mov ebx, 0
(gdb) c
Continuing.
[Inferior 1 (process 17807) exited normally]
(gdb) quit
bsemenov@fedora:~/work/arch-pc/lab09$
```

Рис 23

21. Копируем файл ‘lab8-2.asm’ (из лабораторной работы №8) с программой, выводящей аргументы командной строки, в новый файл ‘lab09-3.asm’. Затем создаём исполняемый файл. Загружаем его в отладчик, передав необходимые аргументы.

```
bsemenov@fedora:~/work/arch-pc/lab09$ gedit lab9-3.asm
bsemenov@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
bsemenov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
bsemenov@fedora:~/work/arch-pc/lab09$ gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Fedora Linux) 16.2-3.fc42
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 lab9-3...
(gdb)
```

Рис 24

22. Устанавливаем точку останова перед первой инструкцией программы и запускаем её выполнение.

```
(gdb) b _start
Breakpoint 1 at 0x8048148: file lab9-3.asm, line 7.
(gdb) run
Starting program: /home/bsemenov/work/arch-pc/lab09/lab9-3 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 lab9-3.asm:7
7       pop ecx
(gdb)
```

Рис 25

23. Адрес вершины стека хранится в регистре ‘esp’, и по этому адресу находится значение, равное количеству аргументов командной строки (включая имя самой программы).

```
(gdb) x/x $esp
0xffffcf90:     0x00000005
(gdb)
```

Рис 26

24. Посмотрим остальные позиции стека – по адесу [esp+4] располагается адрес в памяти, где находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по аресу [esp+12] – второго и т.д.

```
(gdb) x/x $esp
0xffffcf90:     0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffd166:     "/home/bsemenov/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd18f:     "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd1a1:     "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd1b2:     "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd1b4:     "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)
```

Рис 27

32 - битная система = 32 бита на
адрес 1 байт = 8 бит

32 бита / 8 = 4 байта на один указатель
Именно поэтому для доступа к каждому следующему элементу в этом массиве указателей нужно увеличивать адрес на 4 байта.

5 Задания для самостоятельной работы

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

```
%include "in_out.asm"

SECTION .data
msg_func db "Функция: f(x)=17+5x",0
msg_result db "Результат: ",0

SECTION .text
global _start

_start:
mov eax, msg_func
call sprintLF

pop ecx
pop edx
sub ecx,1
mov esi,0

next:
cmp ecx,0h
jz _end
pop eax
call atoi

mov ebx,5
mul ebx
add eax,17

add esi,eax
```

```
add esi,eax
loop next

_end:
mov eax,msg_result
call sprint
mov eax,esi
call iprintLF
call quit
```

Рис 28

```
bsemenov@fedora:~/work/arch-pc/lab09$ gedit lab9-4.asm
bsemenov@fedora:~/work/arch-pc/lab09$ nasm -f elf lab9-4.asm
bsemenov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
bsemenov@fedora:~/work/arch-pc/lab09$ ./lab9-4
Функция: f(x)=17+5x
Результат: 0
bsemenov@fedora:~/work/arch-pc/lab09$
```

Рис 29

2.В листинге 9.3 приведена программа вычисления выражения (3 + 2) ∗ 4 + 5. При запуске данная программа дает неверный результат. Проверим это. С помощью отладчика GDB, анализируя изменения значений регистров, определим ошибку и исправим ее.

lab9-3.asm

```
%include 'in_out.asm'

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

SECTION .text
GLOBAL _start

_start:
    ;---- Вычисление выражения (3+2)*4+5
    mov eax,3       ; eax = 3
    add eax,2       ; eax = 3 + 2 = 5
    mov ebx,4       ; ebx = 4
    mul ebx         ; eax = 5 * 4 = 20
    add eax,5       ; eax = 20 + 5 = 25
    mov edi,eax     ; сохраняем результат

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

Рис 30


bsemenov@fedora:~/work/arch-pc/lab09$ nasm -f elf lab9-5.asm
bsemenov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
bsemenov@fedora:~/work/arch-pc/lab09$ ./lab9-5
Результат: 25
bsemenov@fedora:~/work/arch-pc/lab09$

Рис 31

6 Выводы

Изучены подпрограммы NASM и основы отладки в GDB, включая структурирование кода и поиск ошибок через пошаговое выполнение и анализ регистров.

Список литературы:

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.

2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.

3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander.org/.

4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.

5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 c. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.

6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 c. — ISBN 978-1491941591.

7. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017. 11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.

8. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.

9. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.