

Лабораторная работа №9

Понятие подпрограммы. Отладчик GDB.

Акунаева Антонина Эрдниевна

Содержание

1	Цель работы	6
2	Задание	7
3	Выполнение лабораторной работы	8
3.1	Реализация подпрограмм в NASM	8
3.2	Отладка программ с помощью GDB	12
3.3	Добавление точек останова	18
3.4	Работа с данными программы в GDB	20
3.5	Обработка аргументов командной строки в GDB	25
4	Описание результатов выполнения заданий для самостоятельной работы	29
5	Выводы	37

Список иллюстраций

3.1	Использование команд mkdir и touch	8
3.2	Mcedit: листинг 9.1 в файле lab09-1.asm	9
3.3	Создание и запуск исполняемого файла lab09-1	10
3.4	Mcedit: файл lab09-1.asm: добавление _subcalcul	11
3.5	Создание и запуск исполняемого файла lab09-1: добавление _subcalcul	12
3.6	Создание lab09-2.asm при помощи touch	12
3.7	Mcedit: листинг 9.2 в файле lab09-2.asm	13
3.8	Создание и запуск исполняемого файла lab09-2: добавление -g	14
3.9	Запуск lab09-2 в отладчике GDB	14
3.10	GDB: запуск программы	15
3.11	GDB: установка Breakpoint	15
3.12	GDB: дисассимилированный код программы	16
3.13	GDB: дисассимилированный код на Intel	17
3.14	GDB: псевдографика	18
3.15	Breakpoint check	19
3.16	Установка Breakpoint в GBD	19
3.17	GBD: все Breakpoint	20
3.18	GBD: использование si	20
3.19	GBD: использование info registers	21
3.20	GBD: просмотр значения переменной по имени	21
3.21	GBD: просмотр значения переменной по адресу	22
3.22	GBD: изменение переменной msg1	22
3.23	GBD: изменение переменной msg2	22
3.24	GBD: разные форматы значения регистра	23
3.25	GBD: изменение значения регистра	24
3.26	GBD: continue и quit - выход из отладчика	25
3.27	Создание lab09-3.asm при помощи sr	25
3.28	Создание и запуск исполняемого файла lab09-3	25
3.29	GDB: загрузка исполняемого файла с аргументами	26
3.30	GDB: установка breakpoint и запуск run	26
3.31	GDB: адрес вершины стека в esp и число аргументов с именем программы	27
3.32	GDB: прочие позиции стека	27
4.1	Создание lab09-4.asm при помощи sr	29
4.2	MCedit: lab09-4.asm	30
4.3	Создание и запуск исполняемого файла lab09-4	31

4.4	Использование команды touch - lab09-5.asm	31
4.5	MCedit: lab09-5.asm	32
4.6	GDB: lab09-5	33
4.7	GDB: lab09-5. Добавление брейкпоинта и следование	34
4.8	MCedit: исправленный lab09-5.asm	35
4.9	Создание и запуск исполняемого файла lab09-5	36

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

Научиться реализовывать подпрограммы в NASM.

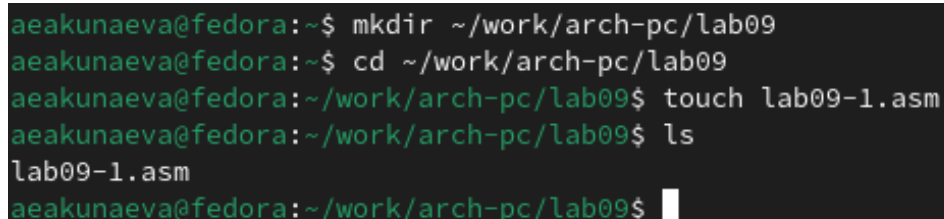
Научиться пользоваться отладчиком в GDB.

Познакомиться с GDB и его возможностями.

3 Выполнение лабораторной работы

3.1 Реализация подпрограмм в NASM

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

A terminal window showing a series of commands and their outputs. The user 'aeakunaeva' is at a 'fedora' machine. They create a directory '~/.work/arch-pc/lab09', change to it, create a file 'lab09-1.asm' using 'touch', and list the directory contents, which shows 'lab09-1.asm'.

```
aeakunaeva@fedora:~$ mkdir ~/.work/arch-pc/lab09
aeakunaeva@fedora:~$ cd ~/.work/arch-pc/lab09
aeakunaeva@fedora:~/.work/arch-pc/lab09$ touch lab09-1.asm
aeakunaeva@fedora:~/.work/arch-pc/lab09$ ls
lab09-1.asm
aeakunaeva@fedora:~/.work/arch-pc/lab09$
```

Рис. 3.1: Использование команд mkdir и touch

Создадим каталог lab09 в рабочем каталоге при помощи mkdir, перейдём в него с cd. В новом каталоге создадим NASM-файл lab09-1.asm при помощи touch.

3.1.2. Внимательно изучите текст программы (Листинг 9.1). Введите в файл lab09-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу.


```

lab09-1.asm      [----]  8 L:[  1+ 6  7/ 37
%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
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер~

```

Рис. 3.2: Mcedit: листинг 9.1 в файле lab09-1.asm

```
aeakunaeva@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
aeakunaeva@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
aeakunaeva@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 10
2x+7=27
aeakunaeva@fedora:~/work/arch-pc/lab09$
```

Рис. 3.3: Создание и запуск исполняемого файла lab09-1

Изучим текст листинга 9.1 - в нём описан текст программы для вычисления выражения $2x+7$ при введённом x с использованием подпрограммы. Скопируем текст из листинга 9.1 в файл lab9-1.asm в текстовом редакторе mcedit, создадим исполняемый файл и запустим его. Программа работает корректно.

Измените 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.

```

lab09-1.asm [----] 21 1
#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
<----->call _subcalcul
<----->mul ebx
<----->add eax, 7
<----->mov [res], eax
<----->ret
<----->
<----->_subcalcul:
<----->    mov ecx, 3
<----->    mul ecx
<----->    sub eax, 1
<----->    ret

```

Рис. 3.4: Mscedit: файл lab09-1.asm: добавление _subcalcul

Впишем подпрограмму подпрограммы `_subcalcul` в `_calcul` для выражения $x = 3x - 1$, которое будет использоваться в $f(x) = 2x + 7$. Также впишем перед умножением на 2 строку-вызов подпрограммы `call _subcalcul`.

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

Рис. 3.5: Создание и запуск исполняемого файла lab09-1: добавление `_subcalcul`

Создадим и запустим новый исполняемый файл и проверим работу. Программа выполняется корректно.

3.2 Отладка программ с помощью GDB

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

```
aeakunaeva@fedora:~/work/arch-pc/lab09$ touch lab09-2.asm
aeakunaeva@fedora:~/work/arch-pc/lab09$ ls
in_out.asm  lab09-1  lab09-1.asm  lab09-1.o  lab09-2.asm
aeakunaeva@fedora:~/work/arch-pc/lab09$
```

Рис. 3.6: Создание `lab09-2.asm` при помощи `touch`

```

lab09-2.asm          [-----] 14
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

```

1По~щъ 2Сохран 3Блок 4Зам

Рис. 3.7: Mcedit: листинг 9.2 в файле lab09-2.asm

Создадим при помощи touch файл lab09-2.asm в текущей директории и впишем туда текст листинга 9.2 в mcedit.

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

```
aeakunaeva@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
aeakunaeva@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
aeakunaeva@fedora:~/work/arch-pc/lab09$ ls
in_out.asm  lab09-1.asm  lab09-2      lab09-2.lst
lab09-1     lab09-1.o    lab09-2.asm  lab09-2.o
aeakunaeva@fedora:~/work/arch-pc/lab09$
```

Рис. 3.8: Создание и запуск исполняемого файла lab09-2: добавление -g

Оттранслируем объектный файл и создадим исполняемый с использованием ключей -l и названия файла для листинга и -g для добавления отладочной информации.

Загрузите исполняемый файл в отладчик gdb.

```
aeakunaeva@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 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) █
```

Рис. 3.9: Запуск lab09-2 в отладчике GDB

Введём gdb lab09-2 для открытия файла в отладчике GDB. Получаем приветственную ознакомительную информацию об отладчике.

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

```
(gdb) run
Starting program: /home/aeakunaeva/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 6730) exited normally]
(gdb) █
```

Рис. 3.10: GDB: запуск программы

Введём команду `run`. Согласимся на использование `debuginfod` в этой сессии, чтобы продолжить. Программа выполняется корректно и выводит надпись “Hello, world!” на экран.

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 9.
(gdb) run
Starting program: /home/aeakunaeva/work/arch-pc/lab09/lab09-2

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

Рис. 3.11: GDB: установка Breakpoint

Установим брейкпоинт в начале программы, в секции `_start`, чтобы подробно рассмотреть её. Запустим и получим обозначение брейкпоинта на указанной метке (строке) и ожидания последующих действий.

Посмотрите дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`.

```

(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) █

```

Рис. 3.12: GDB: дисассимилированный код программы

Введём команду `disassemble` для начальной метки `_start`. Получим дисассимилированный (т.е. переведённый на машинный язык) код нашей программы построчно.

Переключитесь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`.


```

(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) █

```

Рис. 3.13: GDB: дисассимилированный код на Intel

Установим синтаксис, свойственный процессорам Intel, командой `set disassembly-flavor intel`. Повторно ведём команду `disassemble` для начальной метки `_start`. Получим дисассимилированный код, но уже изменённый под синтаксис Intel.

Перечислите различия отображения синтаксиса машинных команд в режимах АТТ и Intel. Включите режим псевдографики для более удобного анализа программы.

```
[ Register Values Unavailable ]

0x8049096      add     BYTE PTR [eax],al
0x8049098      add     BYTE PTR [eax],al
0x804909a      add     BYTE PTR [eax],al
0x804909c      add     BYTE PTR [eax],al
0x804909e      add     BYTE PTR [eax],al
0x80490a0      add     BYTE PTR [eax],al
0x80490a2      add     BYTE PTR [eax],al

native process 6762 In: _start          L9      PC: 0x8049000
(gdb) layout regs
(gdb) layout asm
(gdb) layout regs
(gdb) layout asm
(gdb) layout regs
(gdb)
```

Рис. 3.14: GDB: псевдографика

Синтаксис машинных команд ATТ и Intel различается в:

1. Порядке операндов. В АТТ сначала указывается исходный, а затем результирующий, в Intel - наоборот.
2. Обозначение регистров. В АТТ перед регистрами указывается символ “%”, в Intel может быть “Е” или “R”.
3. Обозначение адресов. В АТТ адреса указываются с использованием скобок и с “\$”, в Intel - без.

3.3 Добавление точек останова

3.3.1. На предыдущих шагах была установлена точка останова по имени метки (_start). Проверьте это с помощью команды `info breakpoints` (кратко `i b`).

```
(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. 3.15: Breakpoint check

Проверим наличие установленной ранее брейкпоинта при помощи `info breakpoints` в терминале. Точка действительно существует на строке 9 в `lab09-2`.

Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определите адрес предпоследней инструкции (`mov ebx,0x0`) и установите точку останова.

```
0x804901b <_start+27>  mov     ebx,0x1
0x8049020 <_start+32>  mov     ecx,0x804a008
0x8049025 <_start+37>  mov     edx,0x7
0x804902a <_start+42>  int     0x80
0x804902c <_start+44>  mov     eax,0x1
b+ 0x8049031 <_start+49>  mov     ebx,0x0
0x8049036 <_start+54>  int     0x80

native process 6762 In: _start          L9      PC: 0x8049000
(gdb) disassembly _start
Undefined command: "disassembly". Try "help".
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
(gdb) disassemble _start
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 22.
(gdb)
```

Рис. 3.16: Установка Breakpoint в GDB

Найдём предпоследнюю инструкцию программы `mov ebx,0x0` с адресом `0x8049031`. Назначим брейкпоинт на эту строку через команду `break *0x8049031` и проверяем в средней полосе отладчика. Точка успешно установлена.

Посмотрите информацию о всех установленных точках останова.

```

native process 6762 In: _start L9 PC: 0x8049000
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 22.
(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:22
(gdb)

```

Рис. 3.17: GDB: все Breakpoint

Введём сокращённую команду `i b` в терминал `gdb` и получим список из двух установленных нами брейкпоинтов.

3.4 Работа с данными программы в GDB

3.4.1. Выполните 5 инструкций с помощью команды `stepi` (или `si`) и проследите за изменением значений регистров. Значения каких регистров изменяются?

```

--Register group: general--
eax      0x8          8
ecx      0x804a000    134520832
edx      0x8          8
ebx      0x1          1
esp      0xffffd0a0   0xffffd0a0
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
0x8049025 <_start+37> mov edx,0x7

native process 6762 In: _start L15 PC: 0x8049016
      breakpoint already hit 1 time
2      breakpoint      keep y   0x08049031 lab09-2.asm:22
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 3.18: GDB: использование `si`

Введём пять раз команду `si` (шаг). Отмечаются активные строки и регистры

записываются в верхней части терминала. Изменяются регистры ebx, ecx, edx, eax, eip.

Посмотреть содержимое регистров также можно с помощью команды `info registers`.

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0a0 0xffffd0a0
ebp      0x0      0x0
esi      0x0      0

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 6762 In: _start L15 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0a0 0xffffd0a0
ebp      0x0      0x0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 3.19: GDB: использование `info registers`

Введём команду `info registers` и получим список регистров и их содержимое в нижней части терминала.

Посмотрите значение переменной `msg1` по имени.

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

Рис. 3.20: GDB: просмотр значения переменной по имени

Введём команду `x/1sb &msg1` и получим информацию о значении переменной `msg1`, обратившись к ней по её имени `msg1`.

Посмотрите значение переменной msg2 по адресу. Посмотрите инструкцию mov ecx,msg2 которая записывает в регистр ecx адрес переменной msg2.

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

Рис. 3.21: GDB: просмотр значения переменной по адресу

Введём команду x/1sb 0x804a008 и получим информацию о значении переменной msg2, обратившись к ней по её адресу, который можно найти в списке дизассемблированной инструкции.

Измените первый символ переменной msg1.

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

Рис. 3.22: GDB: изменение переменной msg1

Заменяем первый символ переменной msg1 = “Hello, world!” H на h командой set {char}&msg1='h' и проверим результат. x/1sb &msg1 показывает нам нужный результат, надпись с маленькой буквы - “hello, world!”.

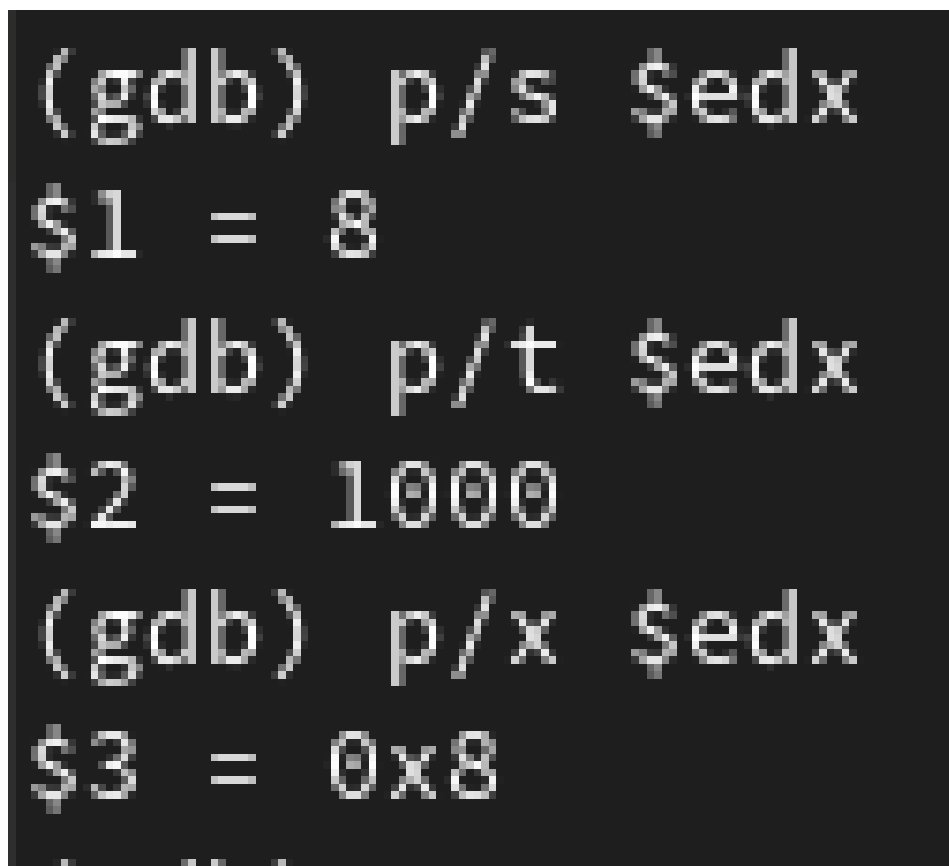
Замените любой символ во второй переменной msg2.

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

Рис. 3.23: GDB: изменение переменной msg2

Тме же образом, но указав в команде `msg2` заменим первую букву фразы на `m` и получим “`world!`” в качестве результата.

Выведите в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра `edx`.



```
(gdb) p/s $edx
$1 = 8

(gdb) p/t $edx
$2 = 1000

(gdb) p/x $edx
$3 = 0x8
```

Рис. 3.24: GDB: разные форматы значения регистра

Просмотрим значения регистра `edx` в различных форматах при помощи `print`: `p/F $edx` (используем форматы `s`, `t`, `x`).

С помощью команды `set` измените значение регистра `ebx`.

```
(gdb) set $ebx='2'  
(gdb) p/s $ebx  
$4 = 50  
(gdb) set $ebx=2  
(gdb) p/s $ebx  
$5 = 2
```

Рис. 3.25: GDB: изменение значения регистра

Изменим значения регистра `ebx` командой `set $ebx='2'` и выведем результат (изменённый регистр) на экран. Затем изменим значение `ebx` на 2 и снова выведем на экран через `print`.

Объясните разницу вывода команд `p/s $ebx`.

Разные значения при выводе объясняются тем, что у значений регистра были разные типы. Так, запись без кавычек присваивает регистру значение 2, а с кавычками - из таблицы ASCII.

Завершите выполнение программы с помощью команды `continue` (сокращенно `c`) или `stepi` (сокращенно `si`) и выйдите из GDB с помощью команды `quit` (сокращенно `q`).


```
(gdb) c
Continuing.
world!

Breakpoint 2, _start () at lab09-2.asm:22
(gdb) q
```

Рис. 3.26: GDB: continue и quit - выход из отладчика

Пропишем команду `c` (continue), чтобы завершить отладку программы, перейдя к её концу и выйдем из отладчика при помощи `q` (quit).

3.5 Обработка аргументов командной строки в GDB

3.5.1. 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`.

```
aeakunaeva@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
aeakunaeva@fedora:~/work/arch-pc/lab09$ ls
image.zip  lab09-1      lab09-1.o  lab09-2.asm  lab09-2.o
in_out.asm lab09-1.asm  lab09-2    lab09-2.lst  lab09-3.asm
aeakunaeva@fedora:~/work/arch-pc/lab09$
```

Рис. 3.27: Создание `lab09-3.asm` при помощи `cp`

Через команду `cp` скопируем `lab8-2.asm` в каталог лабораторной работы 9 под новым названием `lab09-3.asm`, указав его в пункте назначения, относительном пути. Проверим при помощи `ls`.

Создайте исполняемый файл.

```
aeakunaeva@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
aeakunaeva@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
aeakunaeva@fedora:~/work/arch-pc/lab09$ ./lab09-3
aeakunaeva@fedora:~/work/arch-pc/lab09$
```

Рис. 3.28: Создание и запуск исполняемого файла `lab09-3`

Оттранслируем объектный файл и создадим исполняемый с листингом, запустим для проверки.

Загрузите исполняемый файл в отладчик, указав аргументы.

```
aeakunaeva@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2
'аргумент 3'
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 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 lab09-3...
(gdb)
```

Рис. 3.29: GDB: загрузка исполняемого файла с аргументами

Запустим в отладчике gdb файл lab09-3, указав ключ `-args`, чтобы указать аргументы.

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
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 10.
(gdb) run
Starting program: /home/aeakunaeva/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:10
10      pop ecx      ; Извлекаем из стека в `ecx` количество
(gdb) █
```

Рис. 3.30: GDB: установка breakpoint и запуск run

Перед запуском программы отметим брейкпоинт на начале программы в `_start`

и затем запустим через `run`.

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

```
(gdb) x/x $esp
0xffffd060:      0x00000005
(gdb)
```

Рис. 3.31: GDB: адрес вершины стека в `esp` и число аргументов с именем программы

Посмотрите остальные позиции стека – по адресу `[esp+4]` располагается адрес в памяти, где находится имя программы, по адресу `[esp+8]` хранится адрес первого аргумента, по адресу `[esp+12]` – второго и т.д.

```
(gdb) x/x $esp
0xffffd060:      0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd224:      "/home/aeakunaeva/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd250:      "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd262:      "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd273:      "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd275:      "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:      <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 3.32: GDB: прочие позиции стека

Рассмотрим другие позиции стека через обращения к ним в `esp` с шагом 4 и получим верный результат - сначала идёт название программы (имя) в виде адреса, затем по очереди все 4 аргумента.

Объясните, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.).

Шаг изменения адреса равен 4, потому что адресные регистры имеют размерность 4 байта (= 32 бит).

4 Описание результатов выполнения заданий для самостоятельной работы

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

Вариант 13. $f(x)=12x-7$

```
aeakunaeva@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
aeakunaeva@fedora:~/work/arch-pc/lab09$ ls
image.zip  lab09-1.asm  lab09-2.asm  lab09-3      lab09-3.o
in_out.asm lab09-1.o    lab09-2.lst  lab09-3.asm  lab09-4.asm
lab09-1    lab09-2      lab09-2.o    lab09-3.lst
aeakunaeva@fedora:~/work/arch-pc/lab09$
```

Рис. 4.1: Создание lab09-4.asm при помощи cp

Через команду cp скопируем lab8-4.asm с самостоятельной работой 1 из лаб. работы 8 в каталог лабораторной работы 9 под новым названием lab09-4.asm, указав его в пункте назначения, относительном пути. Проверим при помощи ls.

```

lab09-4.asm      [----]  7  L:[  1+32
%include 'in_out.asm'

SECTION .data
msg db "f(x1)+f(x2)+...+f(xn) = ",0

SECTION .text
global _start

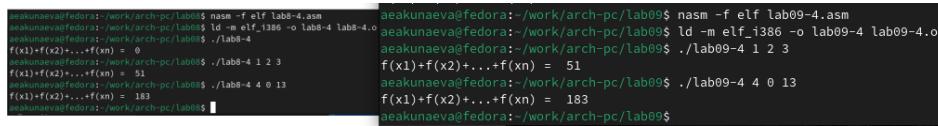
_start:
    pop ecx
    pop edx
    sub ecx,1
    mov esi,0

next:
    cmp ecx,0h
    jz _end
    pop eax
    call atoi
    call _calcul
    add esi,eax
    loop next
_end:
    mov eax,msg
    call sprintf
    mov eax,esi
    call iprintLF
    call quit
_calcul:
    mov ebx,12
    mul ebx
    sub eax,7
    ret

```

Рис. 4.2: MCedit: lab09-4.asm

Откроем файл lab09-4.asm в текстовом редакторе mcedit Midnight Commander и изменим под использование подпрограммы для реализации вычисления функции. Добавим подпрограмму `_calcul` и её вызов в теле `next`.



```

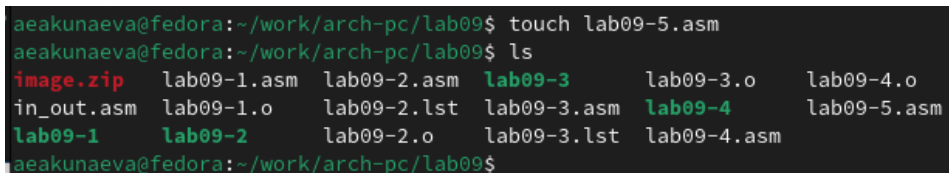
aeakunaeva@fedora: ~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
aeakunaeva@fedora: ~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
aeakunaeva@fedora: ~/work/arch-pc/lab09$ ./lab09-4
f(x1)+f(x2)+...+f(xn) = 0
aeakunaeva@fedora: ~/work/arch-pc/lab09$ ./lab09-4 1 2 3
f(x1)+f(x2)+...+f(xn) = 51
aeakunaeva@fedora: ~/work/arch-pc/lab09$ ./lab09-4 4 0 13
f(x1)+f(x2)+...+f(xn) = 183
aeakunaeva@fedora: ~/work/arch-pc/lab09$
aeakunaeva@fedora: ~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
aeakunaeva@fedora: ~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
aeakunaeva@fedora: ~/work/arch-pc/lab09$ ./lab09-4 1 2 3
f(x1)+f(x2)+...+f(xn) = 51
aeakunaeva@fedora: ~/work/arch-pc/lab09$ ./lab09-4 4 0 13
f(x1)+f(x2)+...+f(xn) = 183
aeakunaeva@fedora: ~/work/arch-pc/lab09$

```

Рис. 4.3: Создание и запуск исполняемого файла lab09-4

Оттранслируем объектный файл и создадим исполняемый для lab09-4.asm. Запустим его, указав аргументы из лабораторной работы 8 ((1,2,3), (4,0,13)) и удостоверимся в корректности работы программы.

4.2. 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.



```

aeakunaeva@fedora: ~/work/arch-pc/lab09$ touch lab09-5.asm
aeakunaeva@fedora: ~/work/arch-pc/lab09$ ls
image.zip  lab09-1.asm  lab09-2.asm  lab09-3      lab09-3.o  lab09-4.o
in_out.asm lab09-1.o    lab09-2.lst  lab09-3.asm  lab09-4    lab09-5.asm
lab09-1    lab09-2      lab09-2.o    lab09-3.lst  lab09-4.asm
aeakunaeva@fedora: ~/work/arch-pc/lab09$

```

Рис. 4.4: Использование команды touch - lab09-5.asm

Создадим NASM-файл lab09-5.asm при помощи touch в текущей директории.

```

lab09-5.asm      [----] 11 L: [ 1+17
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
    mov ebx,3
    mov eax,2
    add ebx,eax
    mov ecx,4
    mul ecx
    add ebx,5
    mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF

call quit

```

Рис. 4.5: MCedit: lab09-5.asm

Откроем файл lab09-5.asm в текстовом редакторе mscedit и впишем туда текст из листинга 9.3.


```

aeakunaeva@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
aeakunaeva@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
aeakunaeva@fedora:~/work/arch-pc/lab09$ ./lab09-5
Результат: 10
aeakunaeva@fedora:~/work/arch-pc/lab09$ gdb lab09-5
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 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 lab09-5...
(No debugging symbols found in lab09-5)
(gdb) run
Starting program: /home/aeakunaeva/work/arch-pc/lab09/lab09-5

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.
Результат: 10
[Inferior 1 (process 10702) exited normally]
(gdb)

```

Рис. 4.6: GDB: lab09-5

Создадим исполняемый файл и проверим работу (выполняется некорректно).
Запустим в отладчике gdb файл lab09-5.

```

Register group: general
eax      0x804a000      134520832
ecx      0x4           4
edx      0x0           0
ebx      0xa           10
esp      0xffffd08c    0xffffd08c
ebp      0x0           0x0
esi      0x0           0
edi      0xa           10
eip      0x804900f      0x804900f <sprint>
eflags   0x10206       [ PF IF RF ]
cs       0x23          35
ss       0x2b          43

>0x804900f <sprint> push edx
0x8049010 <sprint+1> push ecx
0x8049011 <sprint+2> push ebx
0x8049012 <sprint+3> push eax
0x8049013 <sprint+4> call 0x8049000 <slen>
0x8049018 <sprint+9> mov edx,eax
0x804901a <sprint+11> pop eax
0x804901b <sprint+12> mov ecx,eax
0x804901d <sprint+14> mov ebx,0x1
0x8049022 <sprint+19> mov eax,0x4
0x8049027 <sprint+24> int 0x80
0x8049029 <sprint+26> pop ebx
0x804902a <sprint+27> pop ecx

native process 10738 In: sprint L?? PC: 0x804900f
es      0x2b          43
--Type <RET> for more, q to quit, c to continue without paging--
fs      0x0           0
gs      0x0           0
(gdb) i b
Num      Type          Disp Enb Address      What
1        breakpoint     keep y   0x080490e8 <_start>
          breakpoint already hit 1 time
2        breakpoint     keep y   0x080490e8 <_start>
          breakpoint already hit 1 time
(gdb) s
Single stepping until exit from function _start,
which has no line number information.
0x0804900f in sprint ()
(gdb)

```

Рис. 4.7: GDB: lab09-5. Добавление брейкпоинта и следование

Добавим брейкпоинт на начало программы в `_start`, после чего откроем в `layout asm` для удобного наблюдения и запустим при помощи `run`. Осмотрим поэтапно (при помощи команды `si`) каждый шаг.

Заметим, что регистр `ebx` изменился лишь дважды - когда к нему прибавили `eax = 2` (`ebx = 3`) и при прибавлении 5 в `add ebx,5`, что и выводится на экран, хотя остальные операции производились над `eax`.

```

lab09-5.asm      [-----] 14 L: [ 1+14
#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 eax,ebx
    mov ecx,4
    mul ecx
    add eax,5
    mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF

call quit

```

Рис. 4.8: MCedit: исправленный lab09-5.asm

Откроем lab09-5.asm в mscedit и отредактируем текст программы так, чтобы к еах прибавляли ebx (и, соответственно, чтобы 5 также прибавляли к еах и в результат записан был он же).

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

Рис. 4.9: Создание и запуск исполняемого файла lab09-5

Оттранслируем объектный файл и создадим исполняемый для lab09-5.asm. Запустим его и удостоверимся в корректности работы программы.

5 Выводы

Я приобрела навыки написания программ с использованием подпрограмм и познакомилась с методами отладки при помощи GDB и его основными возможностями.