

Лабораторная работа №9

Понятие подпрограммы

Софич Андрей Геннадьевич

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
3.1	Реализация подпрограмм в NASM	7
3.2	Отладка программ с помощью GDB	13
3.3	Добавление точек останова	17
3.4	Работа с данными программы в GDB	18
3.5	Обработка аргументов командной строки в GDB	21
3.6	Задание для самостоятельной работы	22
4	Выводы	30
	Список литературы	31

Список иллюстраций

3.1	Создание каталога	7
3.2	Перемещение по директории	7
3.3	Создание файла	7
3.4	Копирование файла	8
3.5	Редактирование файла	9
3.6	Запуск программы	10
3.7	Редактирование файла	11
3.8	Запуск программы	13
3.9	Создание файла	13
3.10	Редактирование файла	14
3.11	Запуск исполняемого файла	15
3.12	Запуск программы в отладчике	15
3.13	Установка брейкпоинта	16
3.14	Запуск	16
3.15	Диссассимилированный код программы	16
3.16	Отображение с Intel'овским синтаксисом	17
3.17	Точка останова	17
3.18	Установка точки останова	18
3.19	Точки останова	18
3.20	info register	18
3.21	Значение переменной по имени	18
3.22	Значение переменной по адресу	19
3.23	Изменение переменной	19
3.24	Изменение второй переменной	19
3.25	Изменение значений в разные форматы	20
3.26	Изменение значений ebx	20
3.27	Копирование файла	21
3.28	Создание файла	21
3.29	Запуск программы с точкой останова	21
3.30	Регистр esp	22
3.31	Позиции стека	22
3.32	Создание файла	23
3.33	Редактирование файла	23
3.34	Запуск программы	25
3.35	Редактирование файла	26
3.36	Запуск программы в отладчике	27
3.37	Действия в отладчике	28

3.38 Запуск программы	29
---------------------------------	----

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. Задание для самостоятельной работы

3 Выполнение лабораторной работы

3.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения работы №9 (рис. 3.1).

```
[andreysofich@fedora ~]$ mkdir ~/work/arch-pc/lab09
```

Рис. 3.1: Создание каталога

Перехожу в созданную директорию (рис. 3.2).

```
[andreysofich@fedora ~]$ cd ~/work/arch-pc/lab09
```

Рис. 3.2: Перемещение по директории

Создаю файл lab09-1.asm в новом каталоге (рис. 3.3).

```
[andreysofich@fedora lab09]$ touch lab09-1.asm
```

Рис. 3.3: Создание файла

Копирую файл in_out.asm в созданный каталог, так как он понадобится для написания программ (рис. 3.4).

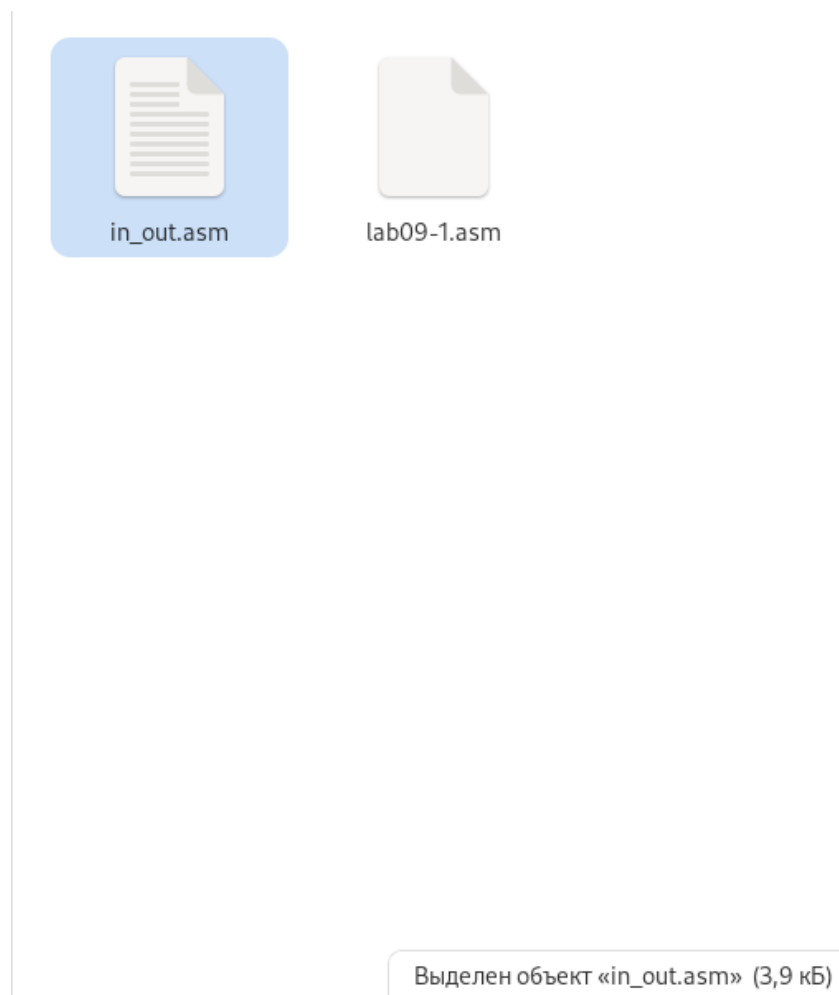


Рис. 3.4: Копирование файла

Открываю файл в GNU nano и переписываю код программы из листинга 9.1 (рис. 3.5).


```

GNU nano 7.2
#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 ; 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

```

Рис. 3.5: Редактирование файла

Создаю объектный файл программы и после компоновки запускаю его (рис. 3.6). Код с подпрограммой работает успешно.

```
[andreysofich@fedora lab09]$ nasm -f elf lab09-1.asm
[andreysofich@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[andreysofich@fedora lab09]$ ./lab09-1
Введите x: 3
2x+7=13
```

Рис. 3.6: Запуск программы

Изменяю текст файла, добавив подпрограмму `sub_calcul` в подпрограмму `_calcul` (рис. 3.7).

```

GNU nano 7.2
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2(3x-1)+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 ; Вызов подпрограммы _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
_calcul:
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы
_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret

```

Рис. 3.7: Редактирование файла

```

#include 'in_out.asm'

SECTION .data

msg: DB 'Введите x: ',0
result: DB '2(3x-1)+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 ; Вызов подпрограммы _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
_calcul:
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы
_subcalcul:
mov ebx, 3
mul ebx

```

```
sub eax,1
ret
```

Запускаю исполняемый файл (рис. 3.8). Программа работает верно.

```
[andreysofich@fedora lab09]$ nasm -f elf lab09-1.asm
[andreysofich@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[andreysofich@fedora lab09]$ ./lab09-1
Введите x: 4
 $2(3x-1)+7=29$ 
```

Рис. 3.8: Запуск программы

3.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm, используя команду touch (рис. 3.9).

```
[andreysofich@fedora lab09]$ touch lab09-2.asm
```

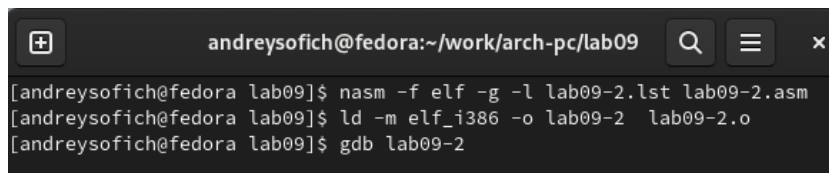
Рис. 3.9: Создание файла

Записываю код программы из листинга 9.2, который выводит сообщение Hello world (рис. 3.10).

```
GNU nano 7.2
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. 3.10: Редактирование файла

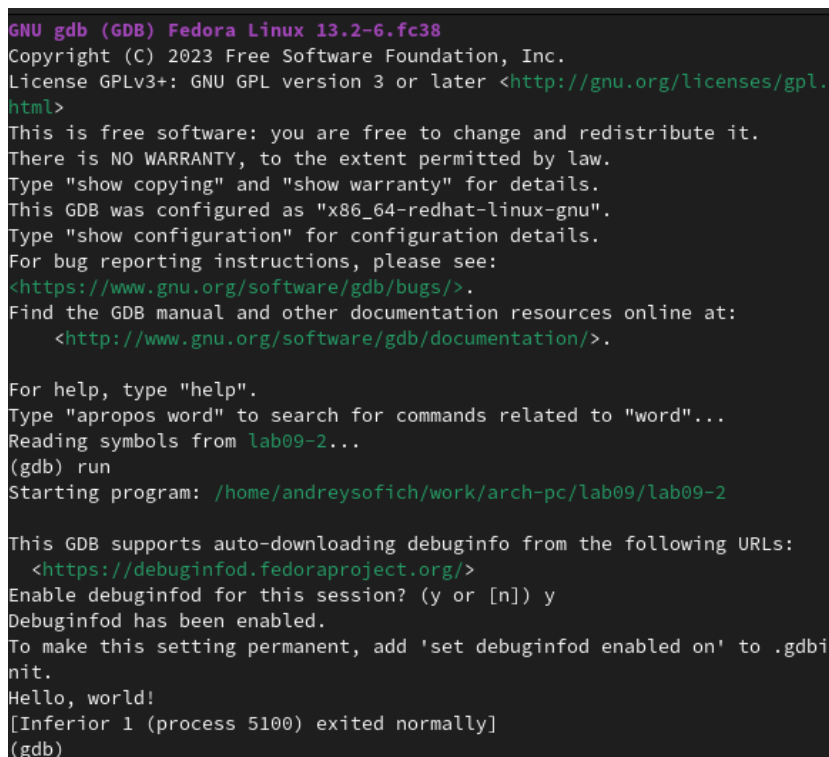
Получаю исполняемый файл. для работы с GDB провожу трансляцию программ с ключом “-g” и загружаю исполняемый файл в отладчик (рис. 3.11).



```
andreysofich@fedora:~/work/arch-pc/lab09
[andreysofich@fedora lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[andreysofich@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
[andreysofich@fedora lab09]$ gdb lab09-2
```

Рис. 3.11: Запуск исполняемого файла

Проверяю работу программы в оболочке GDB с помощью команды run (рис. 3.12).



```
GNU gdb (GDB) Fedora Linux 13.2-6.fc38
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) run
Starting program: /home/andreysofich/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.
Hello, world!
[Inferior 1 (process 5100) exited normally]
(gdb)
```

Рис. 3.12: Запуск программы в отладчике

Для более подробного анализа устанавливаю брейкпоинт на метку _start,с которой начинается выполнение ассемблерной программы (рис. 3.13).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
```

Рис. 3.13: Установка брейкпоинта

Запускаю её (рис. 3.14).

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

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

Рис. 3.14: Запуск

С помощью команды “disassemble _start” просматриваю дисассимилированный код программы (рис. 3.15).

```
(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.
```

Рис. 3.15: Дисассимилированный код программы

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду “set disassembly-flavor intel” (рис. 3.16).


```

(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.

```

Рис. 3.16: Отображение с Intel'овским синтаксисом

Основное различие заключается в том, что в режиме Intel пишется сначала сама команда, а потом её машинный код, в то время как в режиме АТТ идет сначала машинный код, а только потом сама команда.

3.3 Добавление точек останова

Проверяю наличие точки останова с помощью команды `info breakpoints (i b)` (рис. 3.17).

```

(gdb) i b
Num      Type             Disp Enb Address      What
1        breakpoint      keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time

```

Рис. 3.17: Точка останова

Устанавливаю ещё одну точку останова по адресу инструкции, которую можно найти в средней части в левом столбце соответствующей инструкции (рис. 3.18).

```
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
```

Рис. 3.18: Установка точки останова

Просматриваю информацию о точках останова (рис. 3.19).

```
(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
```

Рис. 3.19: Точки останова

3.4 Работа с данными программы в GDB

Просматриваю содержимое регистров с помощью команды info register (i r) (рис. 3.20).

```
eax      0x0          0
ecx      0x0          0
edx      0x0          0
ebx      0x0          0
esp      0xffffd1c0    0xffffd1c0
ebp      0x0          0x0
esi      0x0          0
edi      0x0          0
eip      0x8049000    0x8049000 <_start>
```

Рис. 3.20: info register

Узнаю значение переменной msg1 по имени (рис. 3.21).

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

Рис. 3.21: Значение переменной по имени

Просматриваю значение переменной msg2 по адресу, который можно определить по дизассемблированной инструкции (рис. 3.22).

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

Рис. 3.22: Значение переменной по адресу

Меняю первый символ переменной msg1 (рис. 3.23).

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

Рис. 3.23: Изменение переменной

Также меняю первый символ переменной msg2 (рис. 3.24).

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

Рис. 3.24: Изменение второй переменной

Вывожу значение регистра edx в различных форматах (в шестнадцатеричном, двоичном и символьном форматах) (рис. 3.25).

```

(gdb) p/s $edx
$1 = 7
(gdb) p/t $edx
$2 = 111
(gdb) p/x $edx
$3 = 0x7

```

Рис. 3.25: Изменение значений в разные форматы

С помощью команды set изменяю значение регистра ebx (рис. 3.26).

```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$3 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$4 = 2

```

Рис. 3.26: Изменение значений ebx

Значение регистра отличаются, так как в первом случае мы выводим код символа 2, который в десятичной системе счисления равен 50, а во втором случае выводится число 2, представленное в этой же системе.

3.5 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, который выводит на экран аргументы, в файл с именем lab09-3.asm (рис. 3.27).

```
[andreysofich@fedora lab09]$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
```

Рис. 3.27: Копирование файла

Создаю исполняемый файл, используя ключ `-args` для загрузки программы в GDB. Загружаю исполняемый файл, указав аргументы (рис. 3.28).

```
[andreysofich@fedora lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[andreysofich@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[andreysofich@fedora lab09]$ gdb --args lab09-3 аргумент1 аргумент2 'аргумент3'
GNU gdb (GDB) Fedora Linux 13.2-6.fc38
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...
```

Рис. 3.28: Создание файла

Устанавливаю точку останова перед первой инструкцией в программе и запускаю её (рис. 3.29).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/andreysofich/work/arch-pc/lab09/lab09-3 аргумент1 аргумент2 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:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
```

Рис. 3.29: Запуск программы с точкой останова

Просматриваю адрес вершины стека, который хранится в регистре esp (рис. 3.30).

```
(gdb) x/x $esp
0xffffd170:      0x00000005
```

Рис. 3.30: Регистр esp

Ввожу другие позиции стека- в отличие от адресов, располагается адрес в памяти: имя, первый аргумент, второй и т.д (рис. 3.31).

```
(gdb) x/x $esp
0xffffd170:      0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffd31d:      "/home/andreysofich/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd34b:      "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd35d:      "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd36e:      "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd370:      "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:      <error: Cannot access memory at address 0x0>
```

Рис. 3.31: Позиции стека

Количество аргументов командной строки 4, следовательно и шаг равен четырем.

3.6 Задание для самостоятельной работы

Создаю файл для первого самостоятельного задания, который будет называться lab09-4.asm (рис. 3.32).

```
[andreysofich@fedora lab09]$ touch lab09-4.asm
```

Рис. 3.32: Создание файла

Редактирую код программы lab8-4.asm, добавив подпрограмму, которая вычисляет значения функции $f(x)$ (рис. 3.33).

```
%include 'in_out.asm'
SECTION .data
msg db "Ответ: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
mov esi, 0 ; Используем `esi` для хранения
; промежуточных сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
call _calcul
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
_calcul:
mov ebx,15
mul ebx
sub eax,9
ret
```

Рис. 3.33: Редактирование файла

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

```

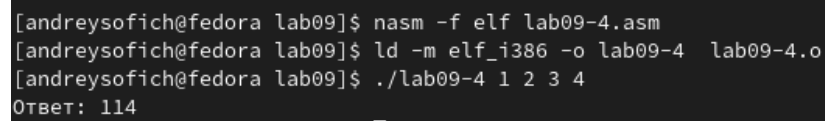
SECTION .data
msg db "Ответ: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
mov esi, 0 ; Используем `esi` для хранения
; промежуточных сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
call _calcul
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы

```



```
_calcul:
mov ebx,15
mul ebx
sub eax,9
ret
```

Создаю исполняемый файл и ввожу аргументы (рис. 3.34). Программа работает верно.



```
[andreysofich@fedora lab09]$ nasm -f elf lab09-4.asm
[andreysofich@fedora lab09]$ ld -m elf_i386 -o lab09-4 lab09-4.o
[andreysofich@fedora lab09]$ ./lab09-4 1 2 3 4
Ответ: 114
```

Рис. 3.34: Запуск программы

Создаю файл и ввожу код из листинга 9.3 (рис. 3.35).

```

GNU nano 7.2                               /home/andre
#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. 3.35: Редактирование файла

Открываю файл в отладчике GDB и запускаю программу (рис. 3.36). Программа выдает ответ 10.

```

[andreysofich@fedora lab09]$ nasm -f elf -g -l lab09.3.lst lab09.3.asm
[andreysofich@fedora lab09]$ ld -m elf_i386 -o lab09.3 lab09.3.o
[andreysofich@fedora lab09]$ gdb lab09.3
GNU gdb (GDB) Fedora Linux 13.2-6.fc38
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) run
Starting program: /home/andreysofich/work/arch-pc/lab09/lab09.3

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.
Результат: 10
[Inferior 1 (process 7072) exited normally]
(gdb)

```

Рис. 3.36: Запуск программы в отладчике

Просматриваю дисассимилированный код программы, ставлю точку останова перед прибавлением 5 и открываю значения регистров на данном этапе (рис. 3.37).

```

The program has no registers now.
(gdb) disassemble _start
Dump of assembler code for function _start:
0x080490e8 <+0>:    mov     $0x3,%ebx
0x080490ed <+5>:    mov     $0x2,%eax
0x080490f2 <+10>:   add     %eax,%ebx
0x080490f4 <+12>:   mov     $0x4,%ecx
0x080490f9 <+17>:   mul     %ecx
0x080490fb <+19>:   add     $0x5,%ebx
0x080490fe <+22>:   mov     %ebx,%edi
0x08049100 <+24>:   mov     $0x804a000,%eax
0x08049105 <+29>:   call    0x804900f <sprint>
0x0804910a <+34>:   mov     %edi,%eax
0x0804910c <+36>:   call    0x8049086 <iprintLF>
0x08049111 <+41>:   call    0x80490db <quit>
End of assembler dump.
(gdb) b *0x080490fb
Breakpoint 1 at 0x80490fb: file lab09.3.asm, line 13.
(gdb) run
Starting program: /home/andreysofich/work/arch-pc/lab09/lab09.3

Breakpoint 1, _start () at lab09.3.asm:13
13      add ebx,5
(gdb) i r
eax            0x8                8
ecx            0x4                4
edx            0x0                0
ebx            0x5                5
esp            0xfffffd190        0xfffffd190
ebp            0x0                0x0
esi            0x0                0
edi            0x0                0
eip            0x80490fb          0x80490fb <_start+19>
eflags         0x206              [ PF IF ]
cs             0x23               35
ss             0x2b               43
ds             0x2b               43
es             0x2b               43
fs             0x0                0
gs             0x0                0

```

Рис. 3.37: Действия в отладчике

Как можно увидеть, регистр `ecx` со значением 4 умножается не на `ebx`, сложенным с `eax`, а только с `eax` со значением 2. Значит нужно поменять значения регистров (например присвоить `eax` значение 3 и просто прибавит 2. После изменений программа будет выглядеть следующим образом:

```

#include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text

```

```

GLOBAL _start

_start:
; ---- Вычисление выражения (3+2)*4+5
mov eax,3
mov ebx,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

```

Попробуем запустить программу (рис. 3.38). Она работает верно.

```

[andreysofich@fedora lab09]$ nasm -f elf lab09.3.asm
[andreysofich@fedora lab09]$ ld -m elf_i386 -o lab09.3 lab09.3.o
[andreysofich@fedora lab09]$ ./lab09.3
Результат: 25

```

Рис. 3.38: Запуск программы

4 Выводы

В данной работе я приобрел навыки написания программ с подпрограммами и познакомился с методами отладки при помощи GDB.

Список литературы

Лабораторная работа №9