

Лабораторная работа №10

Архитектура компьютера

Косолапов Матвей Эдуадович

Содержание

1	Цель работы	6
2	Задание	7
3	Теоретическое введение	8
4	Выполнение лабораторной работы	9
5	Выводы	24
	Список литературы	25

Список иллюстраций

4.1	Создание каталога lab10 и файла lab10-1.asm	9
4.2	Текст программы №1	10
4.3	Ошибка при создании исполняемого файла	10
4.4	Изменение текста программы	11
4.5	Результат выполнения исправленной программы	12
4.6	Новое изменение программы. Добавление подпрограммы _subcalcul	12
4.7	Результат работы программы с подпрограммой _subcalcul	12
4.8	Программа №2	13
4.9	Создание исполняемого файла. Его загрузка в отладчик GDB . . .	14
4.10	Запуск программы в оболочке GDB	14
4.11	Установки точки останова на метку _start. Запуск программы . . .	14
4.12	Дисассимилированный код программы с помощью команды <i>disassemble</i> начиная с метки _start	15
4.13	Дисассимилированный код. Отображение Intel	15
4.14	Включение режим псевдографики	16
4.15	Проверка установленных точек останова	16
4.16	Установка второй точки останова. Просмотр информации о точках останова	16
4.17	Содержание регистров	17
4.18	Содержание регистров после выполнения команды <i>si 5</i>	17
4.19	Значение переменной msg1	17
4.20	Значение переменной msg2	18
4.21	Изменение первого символа строки msg1. Результат изменения .	18
4.22	Изменение первого символа строки msg2. Результат изменения .	18
4.23	Значение регистра <i>edx</i> в разных форматах	18
4.24	Изменение значения регистра <i>ebx</i> на '2'. Результат изменения . .	19
4.25	Изменение значения регистра <i>ebx</i> на 2. Результат изменения . . .	19
4.26	Завершение программы и выход из GDB	19
4.27	Копирование файла lab9-2.asm в файл с именем lab10-3.asm . . .	19
4.28	Создание исполняемого файла, файла листинга lab10-3	20
4.29	Загрузка исполняемого файла в отладчик gdb с аргументами . . .	20
4.30	Установка точки останова перед меткой _start. Запуск программы	20
4.31	Количество аргументов командной строки	21
4.32	Значения располагающиеся в стеке	21
4.33	Код программы	22
4.34	Создание исполняемого файла, проверка его работы	22
4.35	Исправление ошибки	23

4.36 Создание исполняемого файла, проверка его работы	23
---	----

Список таблиц

1 Цель работы

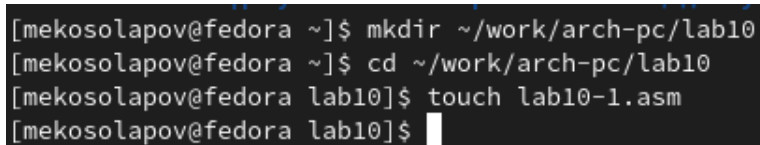
Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Задание

3 Теоретическое введение

4 Выполнение лабораторной работы

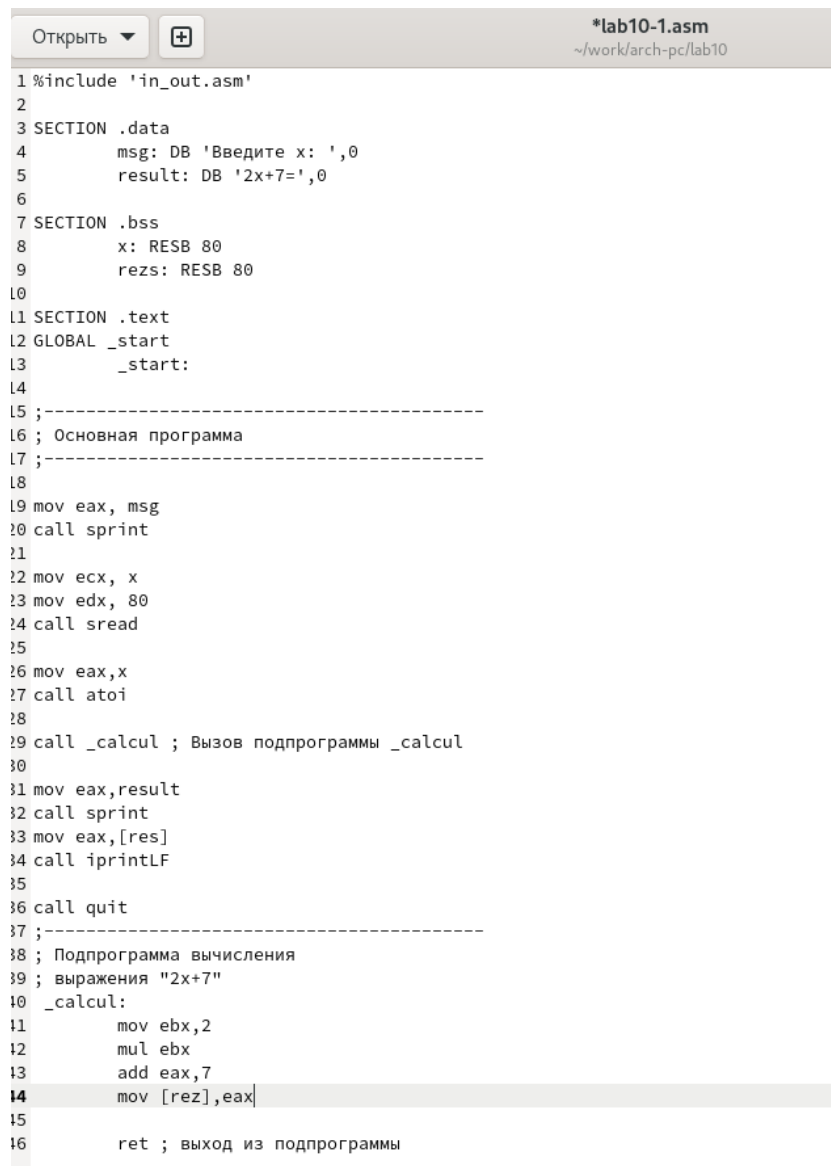
1. Создаём каталог lab10, в нём создаём файл lab10-1.asm(рис. 4.1):



```
[mekosolapov@fedora ~]$ mkdir ~/work/arch-pc/lab10  
[mekosolapov@fedora ~]$ cd ~/work/arch-pc/lab10  
[mekosolapov@fedora lab10]$ touch lab10-1.asm  
[mekosolapov@fedora lab10]$
```

Рис. 4.1: Создание каталога lab10 и файла lab10-1.asm

2. Переносим в файл программу вычисления функции из листинга №1(рис. 4.2):



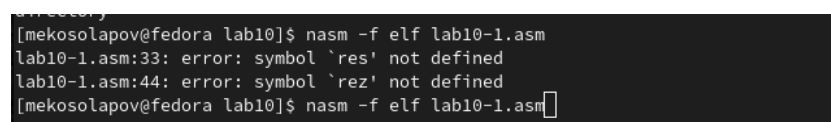
```

1 %include 'in_out.asm'
2
3 SECTION .data
4     msg: DB 'Введите x: ',0
5     result: DB '2x+7=',0
6
7 SECTION .bss
8     x: RESB 80
9     rezs: RESB 80
10
11 SECTION .text
12 GLOBAL _start
13     _start:
14
15 ;-----
16 ; Основная программа
17 ;-----
18
19 mov eax, msg
20 call sprint
21
22 mov ecx, x
23 mov edx, 80
24 call sread
25
26 mov eax,x
27 call atoi
28
29 call _calcul ; Вызов подпрограммы _calcul
30
31 mov eax,result
32 call sprint
33 mov eax,[res]
34 call iprintLF
35
36 call quit
37 ;-----
38 ; Подпрограмма вычисления
39 ; выражения "2x+7"
40 _calcul:
41     mov ebx,2
42     mul ebx
43     add eax,7
44     mov [rez],eax
45
46     ret ; выход из подпрограммы

```

Рис. 4.2: Текст программы №1

3. Создаём исполняемый файл, проверяем работу. Видим ошибку из-за неправильного названия переменных (res,rez,resz).(рис. 4.3):



```

[mekosolapov@fedora lab10]$ nasm -f elf lab10-1.asm
lab10-1.asm:33: error: symbol `res` not defined
lab10-1.asm:44: error: symbol `rez` not defined
[mekosolapov@fedora lab10]$ nasm -f elf lab10-1.asm

```

Рис. 4.3: Ошибка при создании исполняемого файла

4. Исправляем код программы(рис. 4.4):

```

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

```

Рис. 4.4: Изменение текста программы

5. Создаём исполняемый файл, проверяем работу(рис. 4.5):

```
[mekosolapov@fedora lab10]$ nasm -f elf lab10-1.asm
[mekosolapov@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[mekosolapov@fedora lab10]$ ./lab10-1
Введите x: 5
2x+7=17
[mekosolapov@fedora lab10]$
```

Рис. 4.5: Результат выполнения исправленной программы

6. Снова меняем программу, добавляя подпрограмму *_subcalcul*, тоже вычисляющую функцию(рис. 4.6):

```
37 ; -----
38 ; Подпрограмма вычисления
39 ; выражения "2x+7"
40 _calcul:
41     call _subcalcul
42
43     mov ebx,2
44     mul ebx
45     add eax,7
46     mov [res],eax
47
48     ret ; выход из подпрограммы
49 _subcalcul:
50     mov ebx,3
51     mul ebx
52     dec eax
53
54     ret
55
```

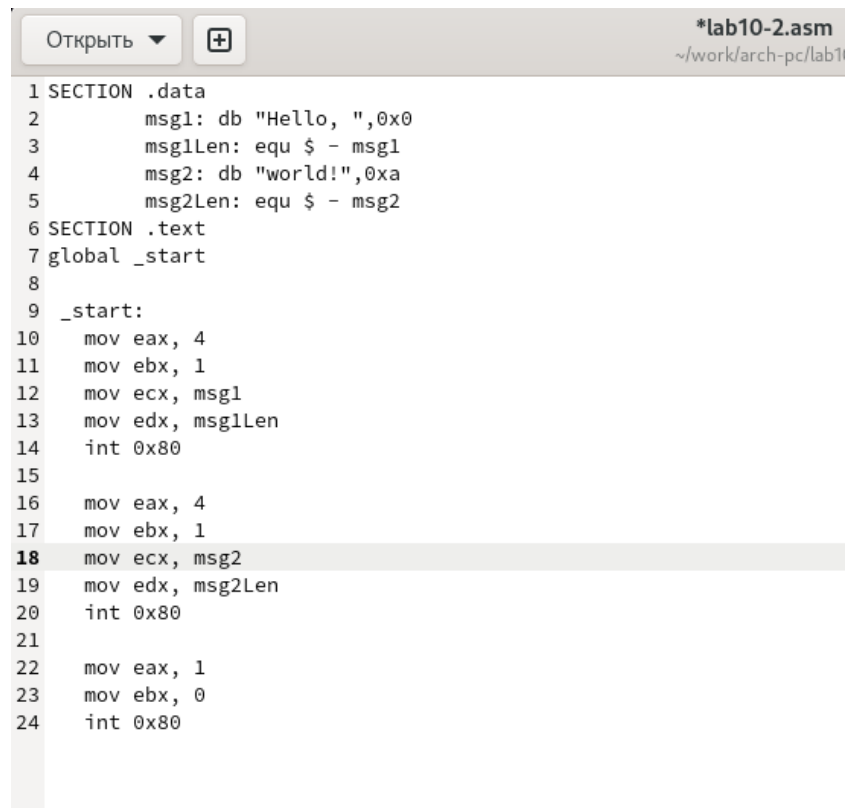
Рис. 4.6: Новое изменение программы. Добавление подпрограммы _subcalcul

7. Создаём исполняемый файл, проверяем работу(рис. 4.7)

```
[mekosolapov@fedora lab10]$ nasm -f elf lab10-1.asm
[mekosolapov@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[mekosolapov@fedora lab10]$ ./lab10-1
Введите x: 5
2x+7=35
[mekosolapov@fedora lab10]$ ./lab10-1
Введите x: 1
2x+7=11
[mekosolapov@fedora lab10]$
```

Рис. 4.7: Результат работы программы с подпрограммой _subcalcul

8. Создаём файл lab10-2.asm и переносим предложенную программу из листинга №2 (рис. 4.8)



```
1 SECTION .data
2     msg1: db "Hello, ",0x0
3     msg1Len: equ $ - msg1
4     msg2: db "world!",0xa
5     msg2Len: equ $ - msg2
6 SECTION .text
7 global _start
8
9 _start:
10    mov eax, 4
11    mov ebx, 1
12    mov ecx, msg1
13    mov edx, msg1Len
14    int 0x80
15
16    mov eax, 4
17    mov ebx, 1
18    mov ecx, msg2
19    mov edx, msg2Len
20    int 0x80
21
22    mov eax, 1
23    mov ebx, 0
24    int 0x80
```

Рис. 4.8: Программа №2

9. Создаём исполняемый файл и файл листинга с ключом -g. Загружаем исполняемый файл в отладчик GDB(рис. 4.9)

```
[mekosolapov@fedora lab10]$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
[mekosolapov@fedora lab10]$ ld -m elf_i386 -o lab10-2 lab10-2.o
[mekosolapov@fedora lab10]$ gdb lab10-2
GNU gdb (GDB) Fedora 11.2-3.fc36
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-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 lab10-2...
(gdb) █
```

Рис. 4.9: Создание исполняемого файла. Его загрузка в отладчик GDB

10. Запускаем программу в оболочке GDB с помощью команды *run*(рис. 4.10):

```
(gdb) run
Starting program: /home/mekosolapov/work/arch-pc/lab10/lab10-2

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 5950) exited normally]
(gdb) █
```

Рис. 4.10: Запуск программы в оболочке GDB

11. Устанавливаем брейкпоинт на метку *_start*. Запускаем программу (рис. 4.11):

```
(gdb) b _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 10.
(gdb) run
Starting program: /home/mekosolapov/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:10
10      mov eax, 4
(gdb) █
```

Рис. 4.11: Установки точки останова на метку *_start*. Запуск программы

12. Смотрим дисассимилированный код программы с помощью команды *disassemble* начиная с метки *_start* (рис. 4.12):

```
(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. 4.12: Дисассимилированный код программы с помощью команды *disassemble* начиная с метки *_start*

13. Переключаемся на Intel'овский синтаксис с помощью команды *set disassembly-flavor intel*. Снова смотрим дисассимилированный код. Видно что в режиме АТТ сначала идут переменные, после основные регистры. А в режиме Intel наоборот (рис. 4.13):

```
(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. 4.13: Дисассимилированный код. Отображение Intel

14. Включаем режим псевдографики(рис. 4.14):

```

[ Register Values Unavailable ]

B> 0x8049000 <_start>    mov     eax,0x4
0x8049005 <_start+5>    mov     ebx,0x1
0x804900a <_start+10>   mov     ecx,0x804a000
0x804900f <_start+15>   mov     edx,0x8
0x8049014 <_start+20>   int     0x80
0x8049016 <_start+22>   mov     eax,0x4
0x804901b <_start+27>   mov     ebx,0x1

native process 5977 In: _start L10 PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 4.14: Включение режим псевдографики

15. Проверяем, установлена ли точка останова с помощью команды *i b* (рис. 4.15):

```

native process 5977 In: _start L10 PC: 0x8049000
(gdb) layout regs
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab10-2.asm:10
          breakpoint already hit 1 time
(gdb)

```

Рис. 4.15: Проверка установленных точек останова

16. Устанавливаем ещё одну точку останова по адресу инструкции `mov eax,0x0`. Смотрим все установленный breakpoints(рис. 4.16):

```

(gdb) b *0x8049036
Breakpoint 2 at 0x8049036: file lab10-2.asm, line 24.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab10-2.asm:10
          breakpoint already hit 1 time
2        breakpoint keep y  0x08049036 lab10-2.asm:24
(gdb)

```

Рис. 4.16: Установка второй точки останова. Просмотр информации о точках останова

17. Смотрим содержание регистров с помощью команды *i r* (рис. 4.17):

```
eflags    0x202      [ IF ]
cs        0x23       35
ss        0x2b       43
ds        0x2b       43
es        0x2b       43
fs        0x0        0
gs        0x0        0
(gdb)
```

Рис. 4.17: Содержание регистров

18. Смотрим содержание регистров после выполнения 5 инструкций с помощью команды *si 5*. Видим, что изменились некоторые регистры. (рис. 4.18):

```
native process 5977 In: _start      L16  PC: 0x8049016
cs        0x23       35             22
ss        0x2b       43
ds        0x2b       43
es        0x2b       43
fs        0x0        0
gs        0x0        0
(gdb) si 5
world!
(gdb)
```

Рис. 4.18: Содержание регистров после выполнения команды *si 5*

19. Смотрим значение переменной *msg1* по имени (рис. 4.19):

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

Рис. 4.19: Значение переменной *msg1*

20. Смотрим значение переменной *msg2* по адресу(рис. 4.20):

```

0x804900f <_start+15> mov     edx,0x8
0x8049014 <_start+20> int     0x80
0x8049016 <_start+22> mov     eax,0x4
0x804901b <_start+27> mov     ebx,0x1
0x8049020 <_start+32> mov     ecx,0x804a008
0x8049025 <_start+37> mov     edx,0x7
0x804902a <_start+42> int     0x80
native process 5977 In: _start L16 PC: 0x8049016
cs      0x23      35      22
gs      0x0       0
(gdb) si 5
world!
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 4.20: Значение переменной msg2

21. Изменяем первый символ переменной msg1 с помощью команды set. Смотрим получившийся результат (рис. 4.21):

```

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

```

Рис. 4.21: Изменение первого символа строки msg1. Результат изменения

22. Так же меняем первый символ переменной msg2. Смотрим результат (рис. 4.22)

```

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

```

Рис. 4.22: Изменение первого символа строки msg2. Результат изменения

23. Выводим в шестнадцатеричном, в двоичном, в символьном форматах значение регистра edx (рис. 4.23)

```

(gdb) p/x $edx
$3 = 0x7
(gdb) p/t $edx
$4 = 111
(gdb) p/s $edx
$5 = 7
(gdb)

```

Рис. 4.23: Значение регистра edx в разных форматах

24. Изменяем значение регистра `ebx` на '2' с помощью команды `set` (рис. 4.24)

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$6 = 50
(gdb)
```

Рис. 4.24: Изменение значения регистра `ebx` на '2'. Результат изменения

25. Изменяем значение регистра `ebx` на 2 с помощью команды `set`. Видим, что выводимый результат отличается. Это потому, что в первом случае мы помещаем в регистр строку '2', а во втором число 2(рис. 4.25):

```
(gdb) set $ebx=2
(gdb) p/s $ebx
$7 = 2
(gdb)
```

Рис. 4.25: Изменение значения регистра `ebx` на 2. Результат изменения

26. Завершаем выполнение программы с помощью команды `c` (рис. 4.26):

```
(gdb) c
Continuing.

Breakpoint 2, _start () at lab10-2.asm:24
(gdb) q
```

Рис. 4.26: Завершение программы и выход из GDB

27. Копируем файл `lab9-2.asm` в файл с именем `lab10-3.asm` (рис. 4.27):

```
[mekosolapov@fedora lab10]$ cp ../lab09/lab9-2.asm lab10-3.asm
[mekosolapov@fedora lab10]$ ls
in_out.asm  lab10-1.asm  lab10-2      lab10-2.lst  lab10-3.asm
lab10-1     lab10-1.o   lab10-2.asm  lab10-2.o
[mekosolapov@fedora lab10]$
```

Рис. 4.27: Копирование файла `lab9-2.asm` в файл с именем `lab10-3.asm`

28. Создаём исполняемый файл и файл листинга с ключом `-g` (рис. 4.28):

```
[mekosolapov@fedora lab10]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
[mekosolapov@fedora lab10]$ ld -m elf_i386 -o lab10-3 lab10-3.o
[mekosolapov@fedora lab10]$
```

Рис. 4.28: Создание исполняемого файла, файла листинга lab10-3

29. Загружаем исполняемый файл в отладчик GDB, указывая аргументы (рис. 4.29):

```
[mekosolapov@fedora lab10]$ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora 11.2-3.fc36
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-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 lab10-3...
(gdb)
```

Рис. 4.29: Загрузка исполняемого файла в отладчик gdb с аргументами

30. Устанавливаем точку останова перед `_start` и запускаем программу (рис. 4.30):

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 5.
(gdb) run
Starting program: /home/mekosolapov/work/arch-pc/lab10/lab10-3 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 lab10-3.asm:5
5      pop ecx      ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 4.30: Установка точки останова перед меткой `_start`. Запуск программы

31. Узнаём количеству аргументов командной строки (рис. 4.31):

```
(gdb) x/x $esp
0xffffd140: 0x00000005
(gdb)
```

Рис. 4.31: Количество аргументов командной строки

32. Смотрим позиции стека по адресам $[esp+4n]$, $n=\{1,2,3,4,5,6\}$. Шаг равен 4, потому что на каждый аргумент выделено 4 байта (рис. 4.32):

```
(gdb) x/x $esp
0xffffd140: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd2fc: "/home/mekosolapov/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd329: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd33b: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd34c: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd34e: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.32: Значения располагающиеся в стеке

Задания для самостоятельной работ 33. Скопируем программу из лабораторной работы №9 и реализуем в ней вычисление функции через подпрограмму (рис. 4.33):

```

1 %include 'in_out.asm'
2 section .data
3 msg1 db 'Функция: f(x) = 4x - 3',0h
4 msg2 db 'Результат: '
5 res dd 0
6 section .text
7 global _start
8 _start:
9 pop ecx
10
11 pop edx
12
13 sub ecx, 1
14 mov eax, msg1
15 call sprintf
16
17 next:
18 cmp ecx, 0h
19 jz _end
20
21 pop eax
22 call atoi
23 call _function
24 loop next
25
26 _end:
27 mov eax, msg2
28 call sprintf
29
30 mov eax, [res]
31 call iprintf
32 call quit
33 _function:
34 ;---Сама функция
35 mov ebx, 4
36 mul ebx
37 add eax, -3
38 add [res],eax
39
40 ret

```

Рис. 4.33: Код программы

34. Создаём исполняемый файл и проверяем его работу (рис. 4.34):

```

[mekosolapov@fedora lab10]$ nasm -f elf myprog.asm
[mekosolapov@fedora lab10]$ ld -m elf_i386 -o myprog myprog.o
[mekosolapov@fedora lab10]$ ./myprog 5 2
Функция: f(x) = 4x - 3
Результат:
22
[mekosolapov@fedora lab10]$ ./myprog 1 2
Функция: f(x) = 4x - 3
Результат:
6

```

Рис. 4.34: Создание исполняемого файла, проверка его работы

35. Теперь найдём ошибку в предложенной программе и исправим её. С помощью gdb отслеживаем работу программы. Исправляем программу для вычисления умножения, поменяв регистры местами (сначала eax умножался на 4, а не ebx) (рис. 4.35):

```

7 ; ---- Вычисление выражения (3+2)*4+5
8     mov ebx,3
9     mov eax,2
10    add eax,ebx
11    mov ecx,4
12    mul ecx
13    add eax,5
14    mov edi,eax
15 ; ---- Вывод результата на экран
16    mov eax,div
17    call sprint
18    mov eax,edi
19    call iprintLF
20    call quit

```

Рис. 4.35: Исправление ошибки

36. Создаём исполняемый файл и проверяем его работу (рис. 4.36):

```

[mekosolapov@fedora lab10]$ nasm -f elf myprog2.asm
[mekosolapov@fedora lab10]$ ld -m elf_i386 -o myprog2 myprog2.o
[mekosolapov@fedora lab10]$ ./myprog2
Результат: 25
[mekosolapov@fedora lab10]$

```

Рис. 4.36: Создание исполняемого файла, проверка его работы

5 Выводы

В ходе данной лабораторной работы я научился работать с программой отладки кода, приобрёл навыки написания программ с использованием подпрограмм.

Список литературы