

Лабораторная работа №10.

Понятие подпрограммы. Отладчик GDB.

Боровиков Даниил Александрович

Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Выводы	22

Список иллюстраций

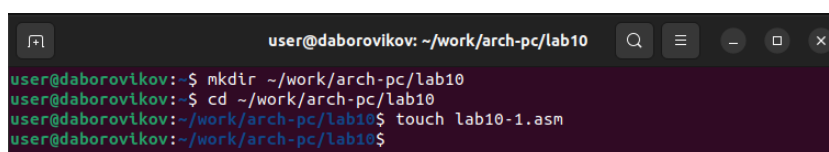
2.1	Создание файла lab10-1.asm в соответствующем каталоге	5
2.2	Текст программы из листинга 10.1.	6
2.3	Запуск исполняемого файла lab10-1.asm	6
2.4	Текст измененной программы	7
2.5	Запуск исправленного исполняемого файла lab10-1.asm	7
2.6	Листинг программы 10.2	8
2.7	Запуск измененного исполняемого файла lab10-2.asm	9
2.8	Дисассимилированный код программы	10
2.9	Режим псевдографики	11
2.10	Брейкпоинты	11
2.11	Дополнительный брейкпоинт	12
2.12	Содержимое регистров	13
2.13	Значения переменных	14
2.14	Замена первого символа msg1	15
2.15	Замена символов msg2	16
2.16	edx в различных форматах	16
2.17	Создание исполняемого файла	17
2.18	Загрузка исполняемого файла в отладчик	17
2.19	Брейкпоинт	17
2.20	Просмотр позиций стека	18

1 Цель работы

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

2 Выполнение лабораторной работы

Создадим каталог для программ лабораторной работы № 10, перейдем в него и создадим файл lab10-1.asm(рис. 2.1)

A screenshot of a terminal window with a dark background. The title bar at the top reads "user@daborovikov: ~/work/arch-pc/lab10". The terminal shows four lines of commands and their outputs: 1. "mkdir ~/work/arch-pc/lab10" followed by a new prompt. 2. "cd ~/work/arch-pc/lab10" followed by a new prompt. 3. "touch lab10-1.asm" followed by a new prompt. 4. A final prompt without a command.

```
user@daborovikov: ~/work/arch-pc/lab10
user@daborovikov:~$ mkdir ~/work/arch-pc/lab10
user@daborovikov:~$ cd ~/work/arch-pc/lab10
user@daborovikov:~/work/arch-pc/lab10$ touch lab10-1.asm
user@daborovikov:~/work/arch-pc/lab10$
```

Рис. 2.1: Создание файла lab10-1.asm в соответствующем каталоге

Введем в файл lab10-1.asm текст программы из листинга 10.1.(рис. 2.2)

```

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

```

Рис. 2.2: Текст программы из листинга 10.1.

Создадим исполняемый файл и запустим его.(рис. 2.3)

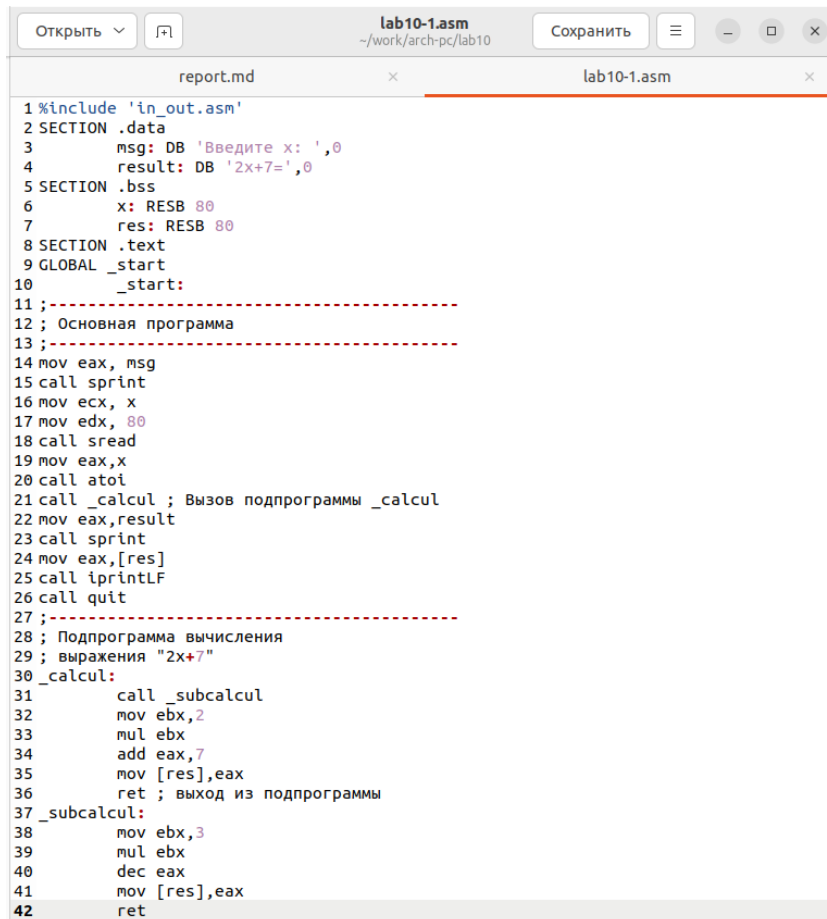
```

user@daborovikov:~/work/arch-pc/lab10$ nasm -f elf lab10-1.asm
user@daborovikov:~/work/arch-pc/lab10$ ld -m elf_i386 -o lab10-1 lab10-1.o
user@daborovikov:~/work/arch-pc/lab10$ ./lab10-1
Введите x: 1
2x+7=9
user@daborovikov:~/work/arch-pc/lab10$

```

Рис. 2.3: Запуск исполняемого файла lab10-1.asm

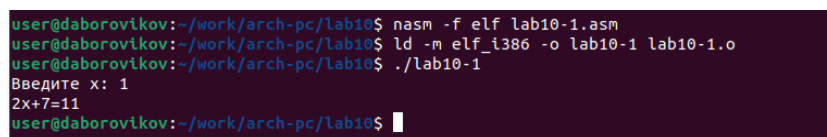
Далее изменим текст программы добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. Т.е. x передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение $g(x)$, результат возвращается в `_calcul` и вычисляется выражение $f(g(x))$. Результат возвращается в основную программу для вывода результата на экран.(рис. 2.4)



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

Рис. 2.4: Текст измененной программы


Создадим исполняемый файл исправленного текста программы lab10-1.asm и запусив его.(рис. 2.5)



```
user@daborovikov:~/work/arch-pc/lab10$ nasm -f elf lab10-1.asm
user@daborovikov:~/work/arch-pc/lab10$ ld -m elf_i386 -o lab10-1 lab10-1.o
user@daborovikov:~/work/arch-pc/lab10$ ./lab10-1
Введите x: 1
2x+7=11
user@daborovikov:~/work/arch-pc/lab10$
```

Рис. 2.5: Запуск исправленного исполняемого файла lab10-1.asm

Создадим файл lab10-2.asm с текстом программы из Листинга 10.2. (Программа печати сообщения Hello world!):(рис. 2.6)



```
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 _start:
9     mov eax, 4
10    mov ebx, 1
11    mov ecx, msg1
12    mov edx, msg1Len
13    int 0x80
14    mov eax, 4
15    mov ebx, 1
16    mov ecx, msg2
17    mov edx, msg2Len
18    int 0x80
19    mov eax, 1
20    mov ebx, 0
21    int 0x80
```

Рис. 2.6: Листинг программы 10.2

Получим исполняемый файл. Для работы с GDB в исполняемый файл добавим отладочную информацию, для этого трансляцию программ проведем с ключом ‘-g’.

```
nasm -f elf -g -l lab10-2.lst lab10-2.asm
```

```
ld -m elf_i386 -o lab10-2 lab10-2.o
```

Загрузим исполняемый файл в отладчик gdb:

```
user@dk4n31:~$ gdb lab10-2
```

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

Для более подробного анализа программы установим брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустим её.(рис. 2.7)


```

user@daborovikov:~/work/arch-pc/lab10$ touch lab10-2.asm
user@daborovikov:~/work/arch-pc/lab10$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
user@daborovikov:~/work/arch-pc/lab10$ ld -m elf_i386 -o lab10-2 lab10-2.o
user@daborovikov:~/work/arch-pc/lab10$ gdb lab10-2
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
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-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) run
Starting program: /home/user/work/arch-pc/lab10/lab10-2
Hello, world!
[Inferior 1 (process 7392) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 9.
(gdb)

```

Рис. 2.7: Запуск измененного исполняемого файла lab10-2.asm

Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`

```
(gdb) disassemble _start
```

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

```
(gdb) set disassembly-flavor intel
```

```
(gdb) disassemble _start(рис. 2.8)
```

```

(gdb) disassemble _start
Dump of assembler code for function _start:
0x08049000 <+0>: mov    $0x4,%eax
0x08049005 <+5>: mov    $0x1,%ebx
0x0804900a <+10>: mov    $0x804a000,%ecx
0x0804900f <+15>: mov    $0x8,%edx
0x08049014 <+20>: int    $0x80
0x08049016 <+22>: mov    $0x4,%eax
0x0804901b <+27>: mov    $0x1,%ebx
0x08049020 <+32>: mov    $0x804a008,%ecx
0x08049025 <+37>: mov    $0x7,%edx
0x0804902a <+42>: int    $0x80
0x0804902c <+44>: mov    $0x1,%eax
0x08049031 <+49>: mov    $0x0,%ebx
0x08049036 <+54>: int    $0x80
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
0x08049000 <+0>: mov    eax,0x4
0x08049005 <+5>: mov    ebx,0x1
0x0804900a <+10>: mov    ecx,0x804a000
0x0804900f <+15>: mov    edx,0x8
0x08049014 <+20>: int    0x80
0x08049016 <+22>: mov    eax,0x4
0x0804901b <+27>: mov    ebx,0x1
0x08049020 <+32>: mov    ecx,0x804a008
0x08049025 <+37>: mov    edx,0x7
0x0804902a <+42>: int    0x80
0x0804902c <+44>: mov    eax,0x1
0x08049031 <+49>: mov    ebx,0x0
0x08049036 <+54>: int    0x80
End of assembler dump.
(gdb) █

```

Рис. 2.8: Дисассимилированный код программы

Синтаксисы машинных команд в режимах АТТ и Intel заключаются в наличии символов “\$” и “&” в АТТ режиме.

Включим режим псевдографики для более удобного анализа программы(рис. 2.9)

```

user@daborovikov: ~/work/arch-pc/lab10

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd2a0 0xffffd2a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+> 0x8049000 <_start> mov $0x4,%eax
0x8049005 <_start+5> mov $0x1,%ebx
0x804900a <_start+10> mov $0x804a000,%ecx
0x804900f <_start+15> mov $0x8,%edx
0x8049014 <_start+20> int $0x80
0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27> mov $0x1,%ebx
0x8049020 <_start+32> mov $0x804a008,%ecx
0x8049025 <_start+37> mov $0x7,%edx
0x804902a <_start+42> int $0x80
0x804902c <_start+44> mov $0x1,%eax
0x8049031 <_start+49> mov $0x0,%ebx
0x8049036 <_start+54> int $0x80
0x8049038 add %al,(%eax)

native process 8652 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) c
The program is not being run.
(gdb) run
Starting program: /home/user/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:9
(gdb)

```

Рис. 2.9: Режим псевдографики

Проверим брейкпоинты командой info breakpoints(рис. 2.10)

```

(gdb) break *<адрес>
A syntax error in expression, near `<адрес>'.
(gdb) i b
Num    Type      Disp Enb Address      What
1      breakpoint keep  y   0x08049000 lab10-2.asm:9
(gdb)

```

Рис. 2.10: Брейкпоинты

Установим еще один брейкпоинт по адресу инструкции и проверим(рис. 2.11)

```

user@daborovikov: ~/work/arch-pc/lab10
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd2a0 0xffffd2a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+ 0x8049000 <_start>    mov    $0x4,%eax
0x8049005 <_start+5>    mov    $0x1,%ebx
0x804900a <_start+10>   mov    $0x804a000,%ecx
0x804900f <_start+15>   mov    $0x8,%edx
0x8049014 <_start+20>   int    $0x80
> 0x8049016 <_start+22> mov    $0x4,%eax
0x804901b <_start+27>   mov    $0x1,%ebx
0x8049020 <_start+32>   mov    $0x804a008,%ecx
0x8049025 <_start+37>   mov    $0x7,%edx
0x804902a <_start+42>   int    $0x80
0x804902c <_start+44>   mov    $0x1,%eax
0x8049031 <_start+49>   mov    $0x0,%ebx

native process 8963 In: _start L14 PC: 0x8049016
info vtbl -- Show the virtual function table for a C++ object.
info warranty -- Various kinds of warranty you do not have.
info watchpoints -- Status of specified watchpoints (all watchpoints if no argument).
info win -- List of all displayed windows.
info xmethod -- GDB command to list registered xmethod matchers.

Type "help info" followed by info subcommand name for full documentation.
Type "apropos word" to search for commands related to "word".
--Type <RET> for more, q to quit, c to continue without paging--Type "apropos -v word" for full documentation of commands related to "word".
Command name abbreviations are allowed if unambiguous.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y 0x08049000 lab10-2.asm:9
breakpoint already hit 1 time
(gdb)

```

Рис. 2.11: Дополнительный брейкпоинт

Выполним 5 инструкций с помощью команды `stepi` (или `si`) и проследите за изменением значений регистров. Посмотрим содержимое регистров с помощью команды `info registers` (рис. 2.12)

```
user@daborovikov: ~/work/arch-pc/lab10

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd2a0 0xffffd2a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+ 0x8049000 <_start>    mov    $0x4,%eax
    0x8049005 <_start+5>  mov    $0x1,%ebx
    0x804900a <_start+10> mov    $0x804a000,%ecx
    0x804900f <_start+15> mov    $0x8,%edx
    0x8049014 <_start+20> int     $0x80
> 0x8049016 <_start+22> mov    $0x4,%eax
    0x804901b <_start+27> mov    $0x1,%ebx
    0x8049020 <_start+32> mov    $0x804a008,%ecx
    0x8049025 <_start+37> mov    $0x7,%edx
    0x804902a <_start+42> int     $0x80
    0x804902c <_start+44> mov    $0x1,%eax
    0x8049031 <_start+49> mov    $0x0,%ebx

native process 8963 In: _start L14 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd2a0 0xffffd2a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(adb)
```

Рис. 2.12: Содержимое регистров

Посмотрим значение переменной msg1 по имени, а msg2 по адресу (рис. 2.13)

```
user@daborovikov: ~/work/arch-pc/lab10

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd2a0 0xffffd2a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+> 0x8049000 <_start> mov $0x4,%eax
0x8049005 <_start+5> mov $0x1,%ebx
0x804900a <_start+10> mov $0x804a000,%ecx
0x804900f <_start+15> mov $0x8,%edx
0x8049014 <_start+20> int $0x80
0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27> mov $0x1,%ebx
0x8049020 <_start+32> mov $0x804a008,%ecx
0x8049025 <_start+37> mov $0x7,%edx
0x804902a <_start+42> int $0x80
0x804902c <_start+44> mov $0x1,%eax
0x8049031 <_start+49> mov $0x0,%ebx
0x8049036 <_start+54> int $0x80
0x8049038 add %al,(%eax)

native process 9989 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) b _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 9.
(gdb) run
Starting program: /home/user/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:9
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)
```

Рис. 2.13: Значения переменных

Изменим первый символ переменной msg1(рис. 2.14)

```
user@daborovikov: ~/work/arch-pc/lab10

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd2a0 0xffffd2a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+> 0x8049000 <_start> mov $0x4,%eax
0x8049005 <_start+5> mov $0x1,%ebx
0x804900a <_start+10> mov $0x804a000,%ecx
0x804900f <_start+15> mov $0x8,%edx
0x8049014 <_start+20> int $0x80
0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27> mov $0x1,%ebx
0x8049020 <_start+32> mov $0x804a008,%ecx
0x8049025 <_start+37> mov $0x7,%edx
0x804902a <_start+42> int $0x80
0x804902c <_start+44> mov $0x1,%eax
0x8049031 <_start+49> mov $0x0,%ebx
0x8049036 <_start+54> int $0x80
0x8049038 add %al,(%eax)

native process 9989 In: _start L9 PC: 0x8049000
Breakpoint 1, _start () at lab10-2.asm:9
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) 
```

Рис. 2.14: Замена первого символа msg1

Изменим символы переменной msg2(рис. 2.15)

```

user@daborovikov: ~/work/arch-pc/lab10

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd2a0 0xffffd2a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+> 0x8049000 <_start> mov $0x4,%eax
0x8049005 <_start+5> mov $0x1,%ebx
0x804900a <_start+10> mov $0x804a000,%ecx
0x804900f <_start+15> mov $0x8,%edx
0x8049014 <_start+20> int $0x80
0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27> mov $0x1,%ebx
0x8049020 <_start+32> mov $0x804a008,%ecx
0x8049025 <_start+37> mov $0x7,%edx
0x804902a <_start+42> int $0x80
0x804902c <_start+44> mov $0x1,%eax
0x8049031 <_start+49> mov $0x0,%ebx
0x8049036 <_start+54> int $0x80
0x8049038 add %al,(%eax)

native process 9989 In: start L9 PC: 0x8049000
'msg1' has unknown type; cast it to its declared type
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}&0x804a008='L'
Attempt to take address of value not located in memory.
(gdb) set {char}&0x804a008='L'
(gdb) set {char}&0x804a00b=' '
(gdb) x/1sb &msg2
0x804a008 <msg2>: "Lor d!\n\034"
(gdb)

```

Рис. 2.15: Замена символов msg2

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.(рис. 2.16)

```

(gdb) p/x $edx
$2 = 0x0
(gdb) p/s $edx
$3 = 0
(gdb) p/t $edx
$4 = 0
(gdb)

```

Рис. 2.16: edx в различных форматах

С помощью команды set изменим значение регистра ebx:(рис. ??)

```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$5 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$6 = 2
(gdb)

```

Разница в командах в

том, что примываем значение числа во втором случае, а в первом символ '2'.

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

Скопируем файл `lab9-2.asm`, созданный при выполнении лабораторной работы №9, с программой выводящей на экран аргументы командной строки (Листинг 9.2) в файл с именем `lab10-3.asm` и создадим исполняемый файл.(рис. 2.17)

```
user@daborovikov:~/work/arch-pc/lab10$ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
user@daborovikov:~/work/arch-pc/lab10$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
user@daborovikov:~/work/arch-pc/lab10$ ld -m elf_i386 -o lab10-3 lab10-3.o
user@daborovikov:~/work/arch-pc/lab10$
```

Рис. 2.17: Создание исполняемого файла

Для загрузки в `gdb` программы с аргументами используем ключ `-args`. Загрузим исполняемый файл в отладчик, указав аргументы(рис. 2.18)

```
user@daborovikov:~/work/arch-pc/lab10$ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
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-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. 2.18: Загрузка исполняемого файла в отладчик

Для начала установим брейкпоинт перед первой инструкцией в программе и запустим ее.(рис. 2.19)

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 5.
(gdb) run
Starting program: /home/user/work/arch-pc/lab10/lab10-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab10-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)
```

Рис. 2.19: Брейкпоинт

Посмотрив позиции стека с шагом +4(рис. 2.20)

```
(gdb) x/x $esp
0xfffffd250: 0x00000005
(gdb) x/s *(void**)($esp + 4)
0xfffffd40e: "/home/user/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)($esp + 8)
0xfffffd434: "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xfffffd446: "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xfffffd457: "2"
(gdb) x/s *(void**)($esp + 20)
0xfffffd459: "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.20: Просмотр позиций стека

Шаг равен размеру переменной - 4 байтам. #Самостоятельная работа

Преобразуем программу из лабораторной работы 9 задание 1 для сам работы,
чтобы вычисления были в подпрограмме(рис. ??)

sam.asm	×	report.md	×
---------	---	-----------	---

```
1 %include 'in_out.asm'
2 SECTION .data
3 fun db "Функция: f(x)=3(X+2)",0
4 msg db "Результат: ",0
5 SECTION .text
6 global _start
7 _start:
8 pop ecx ; Извлекаем из стека в `ecx` количество
9 ; аргументов (первое значение в стеке)
10 pop edx ; Извлекаем из стека в `edx` имя программы
11 ; (второе значение в стеке)
12 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
13 ; аргументов без названия программы)
14 mov esi, 0 ; Используем `esi` для хранения
15 ; промежуточных сумм
16 next:
17 cmp ecx,0h ; проверяем, есть ли еще аргументы
18 jz _end ; если аргументов нет выходим из цикла
19 ; (переход на метку `_end`)
20 pop eax ; иначе извлекаем следующий аргумент из стека
21 call atoi ; преобразуем символ в число
22 call _calc
23 ;add eax,2
24 ;mov ebx,3
25 ;mul ebx
26 add esi,eax
27 ; след. аргумент `esi=esi+eax`
28 loop next ; переход к обработке следующего аргумента
29 _end:
30 mov eax, fun
31 call sprintf
32 mov eax, msg ; вывод сообщения "Результат: "
33 call sprint
34 mov eax, esi ; записываем сумму в регистр `eax`
35 call iprintLF ; печать результата
36 call quit ; завершение программы
37 _calc:
38 add eax,2
39 mov ebx,3
40 mul ebx
41 ret
```

Создадим исполняемый файл для проверки программы(рис. ??)

```
user@daborovikov:~/Рабочий стол$ nasm -f elf sam.asm
user@daborovikov:~/Рабочий стол$ ld -m elf_i386 -o sam sam.o
user@daborovikov:~/Рабочий стол$ ./sam 2
Функция: f(x)=3(X+2)
Результат: 12
user@daborovikov:~/Рабочий стол$ ./sam 2 3
Функция: f(x)=3(X+2)
Результат: 27
user@daborovikov:~/Рабочий стол$
```

Создадим файл для задания 2 сам. работы. И введём программу из листинга 10.3(рис. ??)

```

user@daborovikov: ~/work/arch-pc/lab10
Register group: general
eax 0x8 8
ecx 0x4 4
edx 0x0 0
ebx 0x5 5
esp 0xffffd2a0 0xffffd2a0
ebp 0x0 0x0
esi 0x0 0
edi 0x0 0
eip 0x80490fb 0x80490fb <_start+19>
eflags 0x202 [ IF ]
cs 0x23 35
ss 0x2b 43

B+ 0x80490e8 <_start> mov $0x3,%ebx
0x80490ed <_start+5> mov $0x2,%eax
0x80490f2 <_start+10> add %eax,%ebx
0x80490f4 <_start+12> mov $0x4,%ecx
0x80490f9 <_start+17> mul %ecx
> 0x80490fb <_start+19> add $0x5,%ebx
0x80490fe <_start+22> mov %ebx,%edi
0x8049100 <_start+24> mov $0x804a000,%eax
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov %edi,%eax
0x804910c <_start+36> call 0x8049086 <iprintLF>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 add %al,(%eax)
0x8049118 add %al,(%eax)

native process 12325 In: _start L13 PC: 0x80490fb

Breakpoint 1, _start () at lab10-3.asm:8
(gdb) layout regs
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/user/work/arch-pc/lab10/lab10-3

Breakpoint 1, _start () at lab10-3.asm:8
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Проверим программу с помощью отладчика и найдем ошибку(рис. ??)

```

lab10-3.asm
~/work/arch-pc/lab10
Сохранить

sam.asm x report.md x lab10-3.asm x

1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
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

```

Исправим ошибку(рис. ??)

```
user@daborovikov:~/work/arch-pc/lab10$ gdb lab10-3
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
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-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) r
Starting program: /home/user/work/arch-pc/lab10/lab10-3
Результат: 25
[Inferior 1 (process 12418) exited normally]
(gdb)
```

Запустим рабочую программу(рис. ??)

3 Выводы

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

В ходе лабораторной работы мы приобрели навыки написания программ с использованием подпрограмм, познакомились с методами отладки при помощи GDB и его основными возможностями.

https://github.com/daBorovikov/study_2022-2023_arh-pc-