

Лабораторная работа №10. Понятие подпрограммы. Отладчик GDB.

Дисциплина: Архитектура ЭВМ

Осокин Георгий Иванович НММбд-02-22

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
2.1	Листинг с примером выполнения подпрограммы	7
2.1.1	Исправление листинга	7
2.2	Отладка программы через GDB	10
2.2.1	Точка останова	12
2.2.2	Вывод дисассемблированного кода в формате Intel	12
2.2.3	GDB TUI	14
2.2.4	Точка останова по адресу	14
2.2.5	Просмотр содержимого регистров	15
2.2.6	Изменение значений в памяти	18
2.2.7	Просмотр значений регистров	18
2.2.8	Изменение значений регистров	20
2.2.9	Завершение программы	21
2.3	Обработка аргументов командной строки	21
3	Задания для самостоятельной работы	23
3.1	Вычисление значения функции как подпрограмма	23
3.2	Анализ программы с ошибкой	24
4	Выводы	33

Список иллюстраций

2.1	создавание lab10	7
2.2	Исполнение lab10-1.asm	7
2.3	Ошибки в листинге 10.1	8
2.4	Вывод исправленного lab10-1	9
2.5	Запуск измененного кода lab10-1	10
2.6	Листинг 10.2	11
2.7	Баш скрипт	11
2.8	Первое открытие GDB	12
2.9	Первая точка останова	12
2.10	Код в формате AT&T	13
2.11	Изменение отображения	13
2.12	Режим псевдографики	14
2.13	Новая точка останова	15
2.14	Информация о точках останова	15
2.15	i b	15
2.16	использование stepi 5 раз	16
2.17	Измененные регистры	17
2.18	Просмотр содержимого в переменной	17
2.19	Просмотр содержимого в переменной по адресу	17
2.20	Просмотр содержимого строки по адресу	18
2.21	Изменение переменной	18
2.22	Изменение двух символов	18
2.23	Вывод значения регистра	19
2.24	Вывод значения регистра в разных форматах	20
2.25	Изменение ebx	20
2.26	Продолжение выполнения программы	21
2.27	GDB с флагом аргументов	21
2.28	Точка останова	22
2.29	Вывод значение регистра и аргументов	22
3.1	Остановка перед mov	25
3.2	После mov	26
3.3	Измененная строка	26
3.4	Результат измененный программы после mul	27
3.5	Наблюдение изменений в регистре EBX	28
3.6	Измененный EDI	29
3.7	Замена EBX на EAX	30

3.8	Финальный результат	31
3.9	Исполнение lab10-4	31

Список таблиц

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 выполнения лабораторной работы

```
[giosokin:Code]$ mkdir work/arch-pc/lab10
[giosokin:Code]$ cd work/arch-pc/lab10
[giosokin:lab10]$ touch lab10-1.asm
[giosokin:lab10]$
```

Рис. 2.1: создание lab10

2.1 Листинг с примером выполнение подпрограммы

Скопируем листинг 10.1 и запустим его

```
[giosokin:lab10]$ sh run.sh lab10-1
./lab10-1.asm:30: error: symbol `res' not defined
./lab10-1.asm:42: error: symbol `rez' not defined
[giosokin:lab10]$ sh run.sh lab10-1
Введите x: 2
11
[giosokin:lab10]$
```

Рис. 2.2: Исполнение lab10-1.asm

Видим, что программа выполняется с ошибкой.

2.1.1 Исправление листинга

Найдем ошибки в коде и исправим их.

```

msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
rezs: 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, [result]
    call iprintf
    call quit

;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [result], eax
    ret
; выход из подпрограммы

```

Рис. 2.3: Ошибки в листинге 10.1

Запустим код заново и проверим его


```
work/arch-pc/lab10
> sh run.sh lab10-1
Введите x: 2

11

work/arch-pc/lab10
> sh run.sh lab10-1
Введите x: 3
13

work/arch-pc/lab10
> |
```

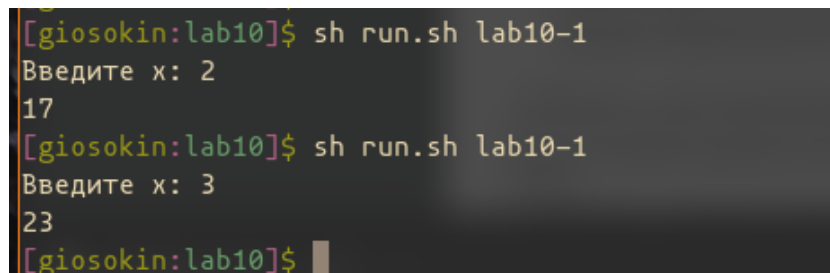
Рис. 2.4: Вывод исправленного lab10-1

Как видим, теперь код выполняется ожидаемо.

Добавим вниз кода подпрограмму

```
34 ; -----
35 ; Подпрограмма вычисления
36 ; выражения "2x+7"
37
38 _calcul:
39     call _subcalcul
40     mov ebx,2
41     mul ebx
42     add eax,7
43     mov [result],eax
44     ret
45 ; выход из подпрограммы
```

Исполним его и посмотрим на результат



```
[giosokin:lab10]$ sh run.sh lab10-1
Введите x: 2
17
[giosokin:lab10]$ sh run.sh lab10-1
Введите x: 3
23
[giosokin:lab10]$
```

Рис. 2.5: Запуск измененного кода lab10-1

2.2 Отладка программы через GDB

Введем в файл lab10-2.asm код из листинга 10.2

```

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. 2.6: Листинг 10.2

Создадим баш скрипт `gen_debug.sh` что б дальше было удобнее работать с созданием исполняемых файлов с отладочной информацией

```

1 gen_debug.sh
1  nasm -f elf -g -l ./${1}.lst ./${1}.asm
1  ld -m elf_i386 -o ./${1} ./${1}.o
~
~
~

```

Рис. 2.7: Баш скрипт

Создадим с помощью этого скрипта исполняемый файл и откроем его через `gdb` введя `gdb lab10-2`

```
[giosokin:lab10]$ gdb lab10-2
GNU gdb (GDB) 12.1
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-unknown-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/horhik/Code/work/arch-pc/lab10/lab10-2
Hello, world!
[Inferior 1 (process 3438) exited normally]
(gdb)
```

Рис. 2.8: Первое открытие GDB

Запустим run и увидим, что код исполнился и вывел Hello world на экран.

2.2.1 Точка останова

Поставим отчку останова на метке _start и запустим программу

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file ./lab10-2.asm, line 13.
(gdb) run
Starting program: /home/horhik/Code/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at ./lab10-2.asm:13
13      mov eax, 4
(gdb)
```

Рис. 2.9: Первая точка останова

Исполнение программы остановилось на метке _start

2.2.2 Вывод дисассемблированного кода в формате Intel

Выведем дизассемблированный код программы

```
Undefined command: "disassemble". Try "help".
(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. 2.10: Код в формате AT&T

Выведенный дизассемблированный код в формате AT&T

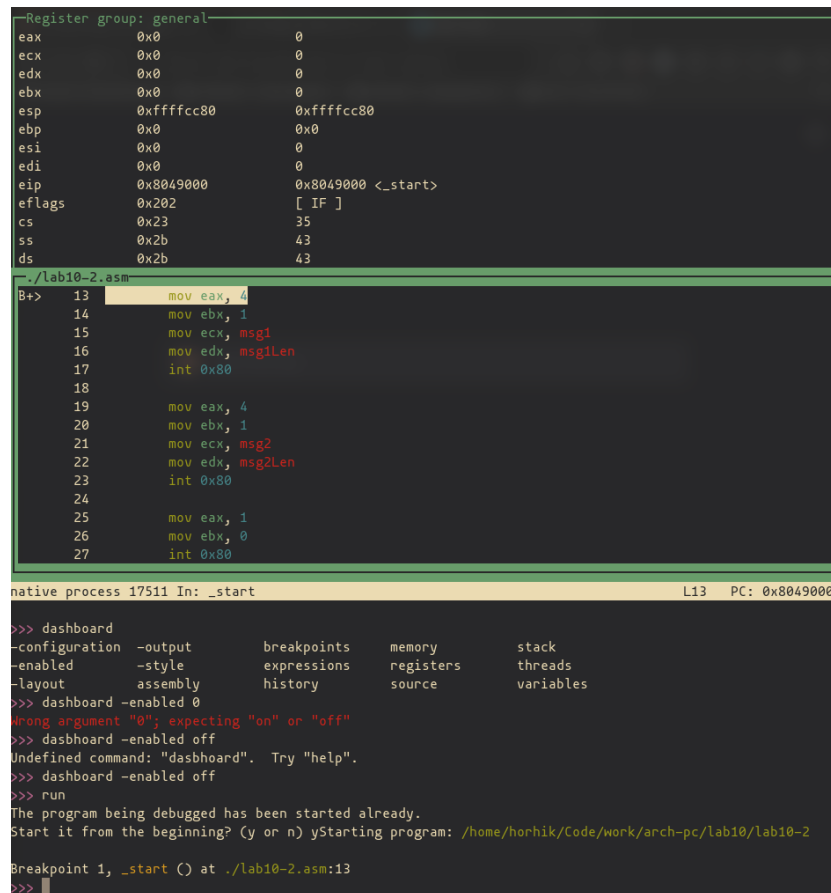
Изменим его на формат отображение от Intel

```
(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. 2.11: Изменение отображения

2.2.3 GDB TUI

Включи режим псевдографики.



```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcc80 0xffffcc80
ebp      0x0      0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43

./lab10-2.asm
B+> 13      mov eax, 4
    14      mov ebx, 1
    15      mov ecx, msg1
    16      mov edx, msg1Len
    17      int 0x80
    18
    19      mov eax, 4
    20      mov ebx, 1
    21      mov ecx, msg2
    22      mov edx, msg2Len
    23      int 0x80
    24
    25      mov eax, 1
    26      mov ebx, 0
    27      int 0x80

native process 17511 In: _start L13 PC: 0x8049000

>>> dashboard
    -configuration  -output      breakpoints  memory      stack
    -enabled        -style      expressions  registers   threads
    -layout         assembly    history     source      variables
>>> dashboard -enabled 0
Wrong argument "0"; expecting "on" or "off"
>>> dashboard -enabled off
Undefined command: "dashboard". Try "help".
>>> dashboard -enabled off
>>> run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/horhik/Code/work/arch-pc/lab10/lab10-2
Breakpoint 1, _start () at ./lab10-2.asm:13
>>>
```

Рис. 2.12: Режим псевдографики

Также, мы можем переключаться между этими режимами, нажимая Ctrl+a
Ctrl+a

2.2.4 Точка останова по адресу

Поставим точку останова на адрес 0x8049031

```

0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax,0x1
b+ 0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x80

exec No process in:
(gdb) layout regs
Undefined command: "layout". Try "help".
(gdb) layout regs
(gdb) break *0x08049031
Breakpoint 2 at 0x08049031: file ./lab10-2.asm, line 26.

```

Рис. 2.13: Новая точка останова

Выведем информацию о поставленных точках останова

```

(gdb) info breakpoints
Num    Type           Disp Enb Address      What
1      breakpoint     keep y   0x08049000  ./lab10-2.asm:13
2      breakpoint     keep y   0x08049031  ./lab10-2.asm:26
(gdb)

```

Рис. 2.14: Информация о точках останова

Можем вывести информацию более короткой командой `i b`

```

(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint     keep y   0x08049000  ./lab10-2.asm:13
(gdb)

```

Рис. 2.15: `i b`

2.2.5 Просмотр содержимого регистров

Запустим программу заново и исполним команду `si` 5 раз, наблюдая как меняются регистры.

```
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
    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
    0x8049038                add     BYTE PTR [eax
    0x804903a                add     BYTE PTR [eax

native process 25839 In: _start
ds          0x2b          43
es          0x2b          43
fs          0x0           0
--Type <RET> for more, q to quit, c to continue w
(gdb) q
A debugging session is active.

        Inferior 1 [process 25839] will be killed

Quit anyway? (y or n) nNot confirmed.
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) █
0 > 0 > gdb > 1 > zsh > 2 > zsh >
```

Рис. 2.16: использование stepi 5 раз

В конце мы видим, что в eax, ebx, ecx и edx уже не нулевые значения.


```
native process 28085 In: _start
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffcc80 0xffffcc80
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
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 2.17: Измененные регистры

Введем `x/1sb &msg1` что бы посмотреть, какие данные лежат в `msg1`

```
Breakpoint 1, _start () at ./lab10-2.asm:13
(gdb) x &msg1
0x804a000 <msg1>:      0x6c6c6548
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb)
```

Рис. 2.18: Просмотр содержимого в переменной

Просмотрим содержимое `&msg2` указав его адрес в памяти

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

Рис. 2.19: Просмотр содержимого в переменной по адресу

Просмотрим содержимое инструкции `mov ecx, msg2`, находящейся по адресу `0x8049020`

```
(gdb) Quit
(gdb) x 0x8049020
0x8049020 <_start+32>:  "\271\b\240\004\b\272\a"
(gdb)
```

Рис. 2.20: Просмотр содержимого строки по адресу

2.2.6 Изменение значений в памяти

Изменим значение в msg1

```
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:  "hello, "
(gdb)
```

Рис. 2.21: Изменение переменной

У нас меняется только первая буква, потому что меняется только первый байт

Заменяем две буквы в переменной &msg2

Мы сдвигаемся на один байт, поэтому +1

```
(gdb) x/1sb &msg2
0x804a008 <msg2>:  "world!\n\034"
(gdb) set {char}0x804a009='0'
(gdb) set {char}0x804a00a='L'
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:  "w0Lld!\n\034"
(gdb)
```

Рис. 2.22: Изменение двух символов

2.2.7 Просмотр значений регистров

Запустим программу заново и посмотрим как меняется регистр eax и выведем значение регистра.

```
eax      0x4      4
edx      0x0      0
ebx      0x0      0
esp      0xffffcd00 0xffffcd00
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43

./lab10-2.asm
B+  13      mov eax, 4
>  14
    15      mov ecx, msg1
    16      mov edx, msg1Len
    17      int 0x80
    18
    19      mov eax, 4
    20      mov ebx, 1
    21      mov ecx, msg2
    22      mov edx, msg2Len
    23      int 0x80
    24
    25      mov eax, 1
    26      mov ebx, 0
    27      int 0x80

native process 6100 In: _start
0x804a000 <msg1>: "Hello, "
(gdb) p/s $eax
$3 = 0
(gdb) p/F $eax
No symbol "F" in current context.
(gdb) p/x $eax
$4 = 0x0
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/horhik/Cod

Breakpoint 1, _start () at ./lab10-2.asm:13
(gdb) si
(gdb) p/s $eax
$5 = 4
(gdb)
```

Рис. 2.23: Вывод значения регистра

Прсмотрим как меняется регистр edx и сделаем вывод в нескольких форматах

```
14      mov ebx, 1
15      mov ecx, msg1
16      mov edx, msg1Len
> 17      int 0x80
18
19      mov eax, 4
20      mov ebx, 1
21      mov ecx, msg2
22      mov edx, msg2Len
23      int 0x80
24
25      mov eax, 1
26      mov ebx, 0
27      int 0x80

native process 6100 In: _start
Start it from the beginning? (y or n) yStarting program: /home/horhik/Code/work/arch-
Breakpoint 1, _start () at ./lab10-2.asm:13
(gdb) si
(gdb) p/s $eax
$5 = 4
(gdb) si
(gdb) si
(gdb) si
(gdb) p/s $edx
$6 = 8
(gdb) p/t $ebx
$7 = 1
(gdb) p/x $edx
$8 = 0x8
(gdb)
```

Рис. 2.24: Вывод значения регистра в разных форматах

2.2.8 Изменение значений регистров

Изменим значение регистра ebx с помощью команды set

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$9 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$10 = 2
(gdb)
```

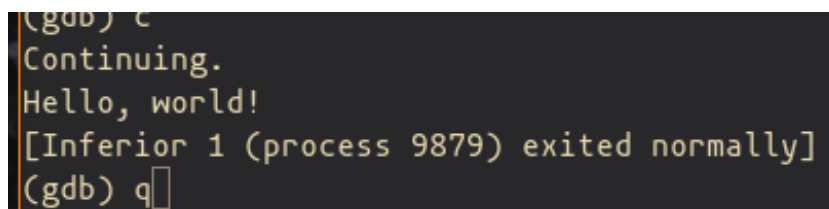
0 > 0 > gdb > 1 > zsh > 2 >

Рис. 2.25: Изменение ebx

Вывод разный, потому что в первом случае мы выводим не число два, а код числа два в таблице ASCII

2.2.9 Завершение программы

Введем с что бы продолжить выполнение программы.



```
(gdb) c
Continuing.
Hello, world!
[Inferior 1 (process 9879) exited normally]
(gdb) q
```

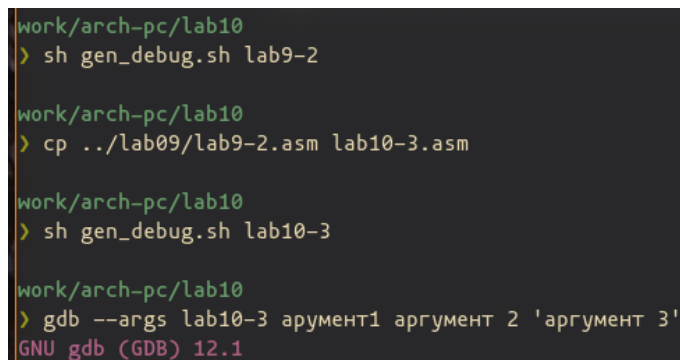
Рис. 2.26: Продолжение выполнения программы

Программа завершила свое выполнение. Можем выйти, нажав q

2.3 Обработка аргументов командной строки

Скопируем файл lab9-2.asm в lab10-4.asm

Откроем этот файл в gdb с флагом аргументов



```
work/arch-pc/lab10
> sh gen_debug.sh lab9-2

work/arch-pc/lab10
> cp ../lab09/lab9-2.asm lab10-3.asm

work/arch-pc/lab10
> sh gen_debug.sh lab10-3

work/arch-pc/lab10
> gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) 12.1
```

Рис. 2.27: GDB с флагом аргументов

Поставим точку останова

```

4  _start:
5
6  _start:
7  pop ecx ; Извлекаем из стека в "ecx" количество
8          ; аргументов (первое значение в стеке)
9  pop edx ; Извлекаем из стека в "edx" имя программы
10         ; (второе значение в стеке)
11  sub ecx, 1 ; Уменьшаем "ecx" на 1 (количество
12            ; аргументов без названия программы)
13  next:
14  cmp ecx, 0 ; проверяем, есть ли еще аргументы
15  jz _end ; если аргументов нет выходим из цикла
16          ; (переход на метку "_end")
17  pop eax ; иначе извлекаем аргумент из стека
18  call sprintf ; вызываем функцию печати

```

```

native process 14544 In: _start
(gdb) b _start
Note: breakpoint 1 also set at pc 0x80490e8.
Breakpoint 2 at 0x80490e8: file ./lab10-3.asm, line 7.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/horhik/Code/work/arch-pc/lab10/
мент1 аргумент 2 аргумент\ 3
Breakpoint 1, _start () at ./lab10-3.asm:7
(gdb) x/x $esp
0xffffccc0: 0x00000005
(gdb)

```

Рис. 2.28: Точка останова

Запустим и выведем значение регистра esp.

После выведем все введенные аргументы

```

(gdb) x/x $esp
0xffffccc0: 0x00000005
0xffffccc4: 0xffffce90
(gdb) x/s *(void**)(esp + 4)
0xffffce90: "/home/horhik/Code/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xffffcebd: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffceed: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffcede: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffcee0: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 2.29: Вывод значение регистра и аргументов

Как видим, когда аргументы закончились, вывелась ошибка

Шаг изменения адреса равен 4, потому что столько места зарезервированно на стеке на указатель введенного аргумента

3 Задания для самостоятельной работы

3.1 Вычисление значения функции как подпрограмма

В 9 лабораторной я уже выделил эту функцию как подпрограмму. Был создан файл `my-function.asm`

```
1    ; f(x) = 17 + 5x
2    ; eax = x
3    ; eax = res
4    magic_function:
5
6    push ebx
7    push ecx
8    push edx
9
10   mov ecx, 5
11   mul ecx
12   add eax, 17
13
14   pop edx
15   pop ecx
16   pop ebx
17
18
```

```
19      ret
```

```
20
```

```
21
```

Вызов этой функции в файле lab9-4.asm

```
30      call atoi ; преобразуем символ в число
```

```
31
```

```
32
```

```
33      call magic_function
```

```
34
```

```
35      add esi, eax ; добавляем к промежуточной сумме
```

```
36
```

3.2 Анализ программы с ошибкой

Запустим код программы через GDB.

Так как умножение происходит в регистр EAX, будем ожидать, что перед умножением в EAX должно лежать $(3+2) = 5$


```
Register group: general
eax    0x2    2
ecx    0x4    4
edx    0x0    0
ebx    0x5    5
esp    0xffffcd00  0xffffcd00
ebp    0x0    0x0
esi    0x0    0
edi    0x0    0
eip    0x80490f9  0x80490f9 <_start+17>
eflags 0x206    [ PF IF ]
cs     0x23    35
ss     0x2b    43
ds     0x2b    43

B+ 0x80490e8 <_start>    mov    ebx,0x3
0x80490ed <_start+5>    mov    eax,0x2
0x80490f2 <_start+10>   add    ebx,eax
0x80490f4 <_start+12>   mov    ecx,0x4
> 0x80490f9 <_start+17> mul    ecx
0x80490fb <_start+19>   add    ebx,0x5
0x80490fe <_start+22>   mov    edi,ebx
0x8049100 <_start+24>   mov    eax,0x804a000
0x8049105 <_start+29>   call   0x804900f <sprint>
0x804910a <_start+34>   mov    eax,edi
0x804910c <_start+36>   call   0x8049086 <iprintLF>
0x8049111 <_start+41>   call   0x80490db <quit>
0x8049116             add    BYTE PTR [eax],al
0x8049118             add    BYTE PTR [eax],al
0x804911a             add    BYTE PTR [eax],al

native process 4099 In: _start
Breakpoint 1, _start () at ./lab10-4.asm:13
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) set disassembly-flavor intel
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/horhik/Code/work/

Breakpoint 1, _start () at ./lab10-4.asm:13
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)
```

Рис. 3.1: Остановка перед mov

Как видим, в EAX не 5

Register group: general		
eax	0x8	8
ecx	0x4	4
edx	0x0	0
ebx	0x5	5
esp	0xffffcd00	0xffffcd00
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

B+	0x80490e8 <_start>	mov	ebx, 0x3
	0x80490ed <_start+5>	mov	eax, 0x2
	0x80490f2 <_start+10>	add	ebx, eax
	0x80490f4 <_start+12>	mov	ecx, 0x4
	0x80490f9 <_start+17>	mul	ecx
>	0x80490fb <_start+19>	add	ebx, 0x5
	0x80490fc <_start+20>	mov	edi, ebx

Рис. 3.2: После mov

$$2 * 4 = 8$$

Вычисление не те, потому что мы перемещаем в EAX 2 и после ничего с ним не делаем.

Мы можем заметить, что нам нужно изменить строчку со сложением esx, eax на eax, esx, что бы результат сложения хранился в EAX.

```
add eax, ebx ; instead ebx, eax
```

Рис. 3.3: Измененная строчка

Запустим заново

eax	0x14	20
ecx	0x4	4
edx	0x0	0
ebx	0x3	3
esp	0xffffcd00	0xffffcd00
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80490fb	0x80490fb <_start+1
eflags	0x206	[PF IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43

B+	0x80490e8	<_start>	mov	\$0x3,%ebx
	0x80490ed	<_start+5>	mov	\$0x2,%eax
	0x80490f2	<_start+10>	add	%ebx,%eax
	0x80490f4	<_start+12>	mov	\$0x4,%ecx
	0x80490f9	<_start+17>	mul	%ecx
>	0x80490fb	<_start+19>	add	\$0x5,%ebx
	0x80490fe	<_start+22>	mov	%ebx,%edi

Рис. 3.4: Результат измененный программы после mul

Как видим, теперь результат верный

Продолжим дебажить.

```
Debugging with gdb - gdb Commands - Clang++ - Linux
Register group: general
eax      0x14      20
ecx      0x4       4
edx      0x0       0
ebx      0x8       8
esp      0xffffcd00 0xffffcd00
ebp      0x0       0
esi      0x0       0
edi      0x0       0
eip      0x80490fe 0x80490fe <_start+22>
eflags   0x202     [ IF ]
cs       0x23      35
ss       0x2b      43
ds       0x2b      43

B+ 0x80490e8 <_start>    mov $0x3,%ebx
0x80490ed <_start+5>    mov $0x2,%eax
0x80490f2 <_start+10>   add %ebx,%eax
0x80490f4 <_start+12>   mov $0x4,%ecx
B+ 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>
b+ 0x804910a <_start+34> mov %edi,%eax
0x804910c <_start+36>   call 0x8049086 <iprintf>
b+ 0x8049111 <_start+41> call 0x80490db <quit>
0x8049116      add %al,%eax
0x8049118      add %al,%eax
0x804911a      add %al,%eax

native process 26736 In: _start L19

Breakpoint 6, _start () at ./lab10-4.asm:26
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/horhik/Code/work/arch-pc/

Breakpoint 1, _start () at ./lab10-4.asm:13
(gdb) si
(gdb) si
(gdb) si
(gdb) si

Breakpoint 4, _start () at ./lab10-4.asm:17
(gdb) si
(gdb) si
(gdb) si
```

Рис. 3.5: Наблюдение изменений в регистре EBX

Как мы видим, число 5 добавилось в регистр EBX, а не в EAX.

А после это значение переносится в EDI. Обычно этот регистр отвечает за ввод.

```
esi      0x0      0
edi      0x8      8
eip      0x8049100 0x8049100 <_start+24>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43

B+ 0x80490e8 <_start>    mov $0x3,%ebx
0x80490ed <_start+5>    mov $0x2,%eax
0x80490f2 <_start+10>   add %ebx,%eax
0x80490f4 <_start+12>   mov $0x4,%ecx
B+ 0x80490f9 <_start+17> mul %ecx
0x80490fb <_start+19>   add $0x5,%ebx
0x80490fe <_start+22>   mov %ebx,%edi
> 0x8049100 <_start+24> mov $0x0,%eax
0x8049105 <_start+29>   call 0x804900f <sprint>
b+ 0x804910a <_start+34> mov %edi,%eax
0x804910c <_start+36>   call 0x8049086 <iprintLF>
b+ 0x8049111 <_start+41> call 0x80490db <quit>
0x8049116          add %al,(%eax)
0x8049118          add %al,(%eax)
0x804911a          add %al,(%eax)

native process 26736 In: _start L22
```

Рис. 3.6: Измененный EDI

После Содержимое EDI выводится на экран, поэтому следует предположить, что добавление 5 к регистру EBX, а не к EAX - не то поведение программы, которое мы ожидаем.

Изменим программу, заменим ebx на eax.

```
add eax, ebx ; Instead  
mov ecx, 4  
mul ecx  
add eax, 5  
mov edi, eax  
  
mov eax, div  
call sprint  
mov eax, edi  
call iprintLF  
call quit
```

Рис. 3.7: Замена EBX на EAX

Запустим программу заново и посмотрим как теперь изменилось ее поведение.

Исполним `continue`, пропустив все точки останова. На экран выводится 25 - ожидаемый результат

```
Register group: general
eax      0x19      25
ecx      0x4
edx      0x0
ebx      0x3
esp      0xffffcd00 0xffffcd00
ebp      0x0
esi      0x0
edi      0x19      25
eip      0x804910c 0x804910c <_start+36>
eflags   0x206     [ PF IF ]
cs       0x23      35
ss       0x2b      43
ds       0x2b      43
es       0x2b      43
fs       0x0
gs       0x0

native No process in:
(gdb) b _start
Breakpoint 1 at 0x80490e8: file ./lab10-4.asm, line 13.
(gdb) b *(_start+36)
Breakpoint 2 at 0x804910c: file ./lab10-4.asm, line 25.
(gdb) run
Starting program: /home/horhik/Code/work/arch-pc/lab10/lab10-4

Breakpoint 1, _start () at ./lab10-4.asm:13
(gdb) c
Continuing.
Результат:
Breakpoint 2, _start () at ./lab10-4.asm:25
(gdb) c
Continuing.
25
(gdb) q 1 (process 31939) exited normally]
```

Рис. 3.8: Финальный результат

Теперь запустим программу не через GDB

```
work/arch-pc/lab10 took 52s
+ > ./lab10-4
Результат: 25
work/arch-pc/lab10
+ >
```

Рис. 3.9: Исполнение lab10-4

Вывелся правильный результат.

4 Выводы

Мы приобрели навыки написания программ с использованием подпрограмм а также ознакомились с процессом отладки через программу GDB и научились пользоваться его основными возможностями.