

# **Отчет по лабораторной работе №10**

*дисциплина: Архитектура компьютера*

Галацан Николай, НПИбд-01-22

# Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Выполнение заданий для самостоятельной работы	17
4	Выводы	26

# Список иллюстраций

2.1	Редактирование файла lab10-1.asm . . . . .	6
2.2	Запуск исполняемого файла lab 10-1 . . . . .	6
2.3	Добавление подпрограммы в lab10-1.asm . . . . .	7
2.4	Запуск измененного исполняемого файла lab 10-1 . . . . .	7
2.5	Редактирование файла lab10-2.asm . . . . .	8
2.6	Запуск исполняемого файла lab10-2 в отладчике . . . . .	8
2.7	Установка брейкпоинта и запуск lab10-2 . . . . .	9
2.8	Промотр дисассимилированного кода программы lab10-2 . . . . .	9
2.9	Режим псевдографики . . . . .	10
2.10	Информация о точках останова. Установка новой по адресу . . . . .	11
2.11	Выполнение инструкций stepi . . . . .	12
2.12	Применение команды info registers . . . . .	12
2.13	Применение команды x для просмотра значений переменных . . . . .	13
2.14	Применение команды set . . . . .	13
2.15	Применение команды r в различных форматах . . . . .	14
2.16	Изменение значения регистра ebx. Просмотр . . . . .	14
2.17	Создание файла lab10-3.asm. Загрузка в отладчик . . . . .	15
2.18	Установка брейкпоинта и запуск lab10-3. Вывод количества аргументов . . . . .	16
2.19	Просмотр остальных позиций стека . . . . .	16
3.1	Запуск исполняемого файла lab10-sam-1 . . . . .	19
3.2	Запуск исполняемого файла lab10-sam-2 . . . . .	19
3.3	Загрузка в отладчик lab10-sam-2 . . . . .	20
3.4	Установка брейкпоинта в lab10-sam-2 и просмотр. Запуск программы . . . . .	21
3.5	Отслеживание значений регистров при вычислении выражения (1) . . . . .	22
3.6	Отслеживание значений регистров при вычислении выражения (2) . . . . .	23
3.7	Отладка измененной программы lab10-sam-2, вывод результата . . . . .	25

# 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

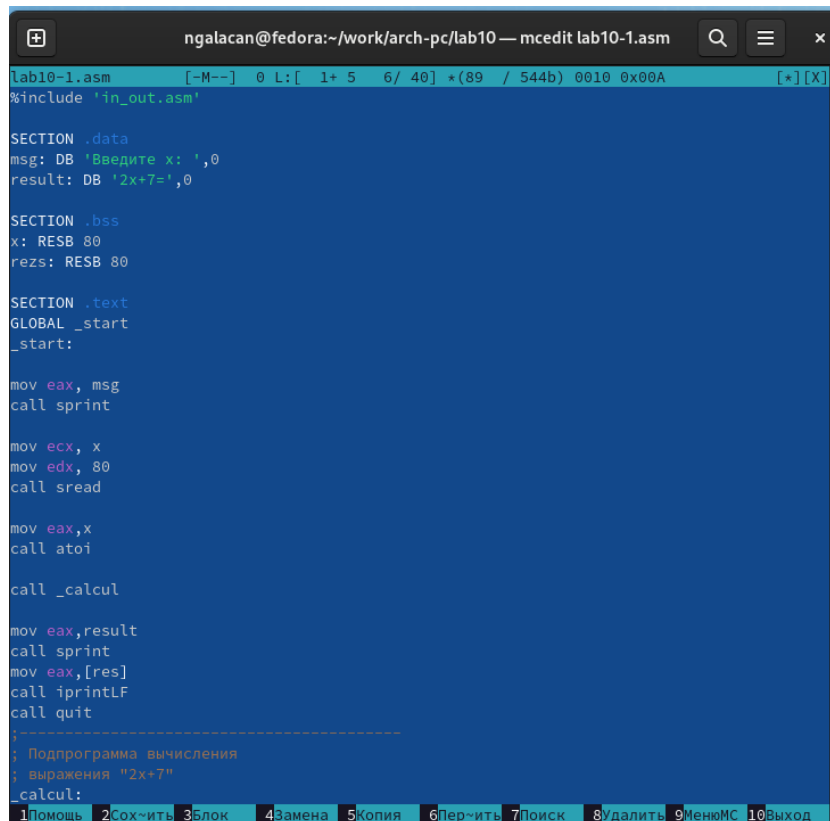
1. Ввожу команды для создания каталога лабораторной работы, перехожу в него, создаю файл `lab10-1.asm`

```
mkdir ~/work/arch-pc/lab10
```

```
cd ~/work/arch-pc/lab10
```

```
touch lab10-1.asm
```

2. Ввожу в файл `lab10-1.asm` текст программы из листинга 10.1, сохраняю файл. (рис. 2.1).



```
lab10-1.asm [-M--] 0 L: [ 1+ 5 6/ 40] *(89 / 544b) 0010 0x00A [*][X]
#include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0

SECTION .bss
x: RESB 80
rez: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax, msg
call sprint

mov ecx, x
mov edx, 80
call sread

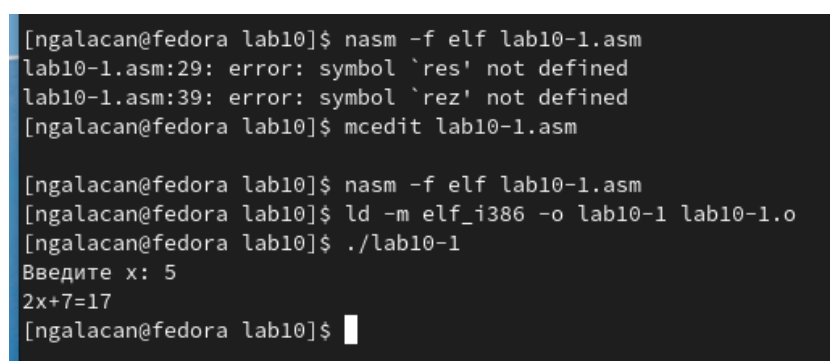
mov eax, x
call atoi

call _calcul

mov eax, result
call sprint
mov eax, [rez]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
```

Рис. 2.1: Редактирование файла lab10-1.asm

Создаю исполняемый файл и запускаю его, предварительно скопировав файл `in_out.asm` в соответствующий каталог. Выводится сообщение об ошибке. Исправляю опечатки в имени переменной и вновь запускаю программу. Программа работает верно (рис. 2.2).



```
[ngalacan@fedora lab10]$ nasm -f elf lab10-1.asm
lab10-1.asm:29: error: symbol `res' not defined
lab10-1.asm:39: error: symbol `rez' not defined
[ngalacan@fedora lab10]$ mcedit lab10-1.asm

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

Рис. 2.2: Запуск исполняемого файла lab 10-1

Далее изменяю текст программы добавив подпрограмму `_subcalcul` в `_calcul`

для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры. (рис. 2.3).

```
; выражения "2x+7"
_calcul:
call _subcalcul
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы
; 3x-1
_subcalcul:
mov ebx,3
mul ebx
sub eax,1
ret
```

Рис. 2.3: Добавление подпрограммы в lab10-1.asm

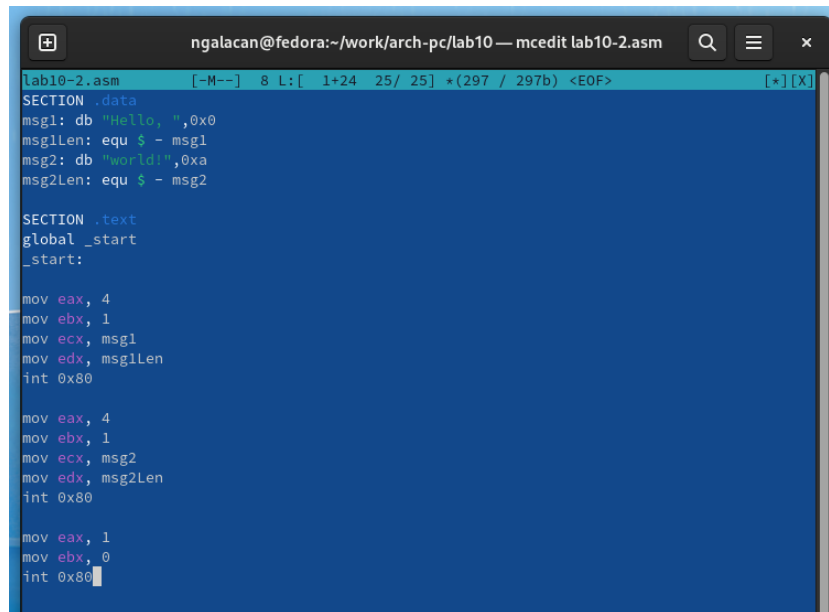
Создаю исполняемый файл и запускаю его (рис. 2.4).

```
[ngalacan@fedora lab10]$ nasm -f elf lab10-1.asm
[ngalacan@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[ngalacan@fedora lab10]$ ./lab10-1
Введите x: 5
f(x)=2x+7, g(x)=3x-1, f(g(x))=35
[ngalacan@fedora lab10]$ ./lab10-1
Введите x: 0
f(x)=2x+7, g(x)=3x-1, f(g(x))=5
[ngalacan@fedora lab10]$ ./lab10-1
Введите x: 1
f(x)=2x+7, g(x)=3x-1, f(g(x))=11
[ngalacan@fedora lab10]$
```

Рис. 2.4: Запуск измененного исполняемого файла lab 10-1

Убеждаюсь в том, что программа вычисляет верное значение.

Создаю новый файл: `touch lab10-2.asm`. Ввожу в него текст программы из листинга 10.2 (рис. 2.5).



```
lab10-2.asm [-M--] 8 L: [ 1+24 25/ 25] *(297 / 297b) <EOF> [*][X]
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

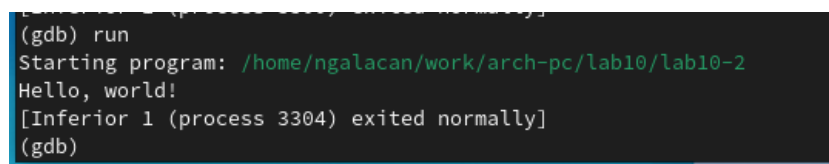
mov eax, 1
mov ebx, 0
int 0x80
```

Рис. 2.5: Редактирование файла lab10-2.asm

Создаю исполняемый файл, с которым можно работать в GDB и загружаю в отладчик:

```
nasm -f elf -g -l lab10-2.lst lab10-2.asm
ld -m elf_i386 -o lab10-2 lab10-2.o
gdb lab10-2
```

Запускаю программу в отладчике (рис. 2.6).



```
(gdb) run
Starting program: /home/ngalacan/work/arch-pc/lab10/lab10-2
Hello, world!
[Inferior 1 (process 3304) exited normally]
(gdb)
```

Рис. 2.6: Запуск исполняемого файла lab10-2 в отладчике

Устанавливаю брейкпоинт на метку \_start и запускаю программу (рис. 2.7).



```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 11.
(gdb) run
Starting program: /home/ngalacan/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:11
11      mov eax, 4
```

Рис. 2.7: Установка брейкпоинта и запуск lab10-2

Просматриваю дисассимилированный код программы начиная с `_start`. Переключаюсь на отображение команд с синтаксисом Intel (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: Промотр дисассимилированного кода программы lab10-2

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel заключаются в том, что в режиме Intel регистры и их значения отображаются более привычно и визуально удобно (колонка справа). Сначала указано назва-

ние регистра, через запятую - его значение. В режиме АТТ наоборот: сначала \$значение, после - %регистр.

Включаю режим псевдографики (рис. 2.9).

```
layout asm
```

```
layout regs
```

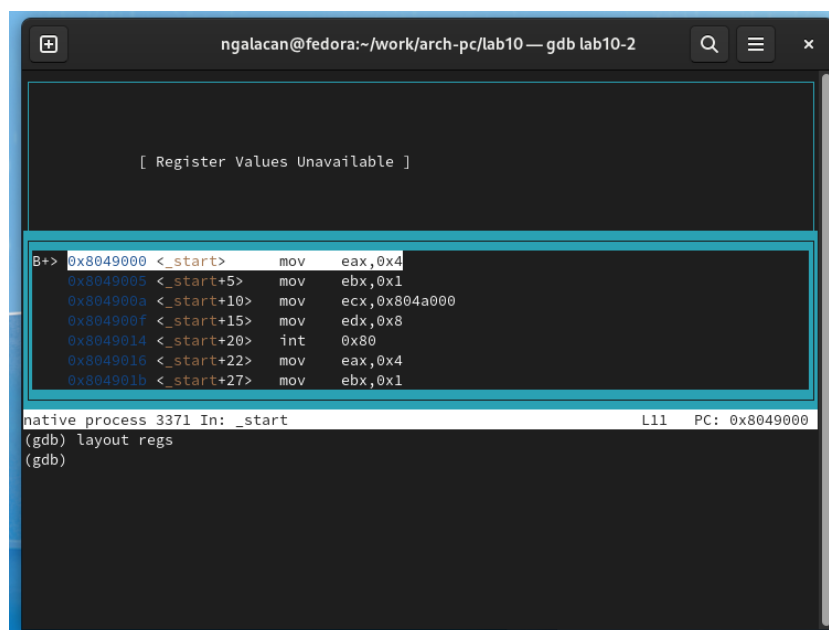


Рис. 2.9: Режим псевдографики

Проверяю наличие установленной точки останова. Устанавливаю еще одну по адресу инструкции, вновь запрашиваю информацию краткой командой (рис. 2.10)

```
ngalacan@fedora:~/work/arch-pc/lab10 — gdb lab10-2  Q  ≡  x

[ Register Values Unavailable ]

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],al
0x804903a add BYTE PTR [eax],al
0x804903c add BYTE PTR [eax],al
0x804903e add BYTE PTR [eax],al
0x8049040 add BYTE PTR [eax],al
0x8049042 add BYTE PTR [eax],al

native process 3836 In: _start L11 PC: 0x8049000
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab10-2.asm:11
breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab10-2.asm, line 24.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab10-2.asm:11
breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab10-2.asm:24
(gdb) 
```

Рис. 2.10: Информация о точках останова. Установка новой по адресу

Выполняю 5 инструкций stepi (рис. 2.11).

```

ngalacan@fedora:~/work/arch-pc/lab10 — gdb lab10-2
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>

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

native process 3836 In: _start L17 PC: 0x8049016
Breakpoint 2 at 0x8049031: file lab10-2.asm, line 24.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y 0x08049000 lab10-2.asm:11
          breakpoint already hit 1 time
2        breakpoint keep y 0x08049031 lab10-2.asm:24
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 2.11: Выполнение инструкций stepi

Поочередно меняются значения регистров eax, ebx, ecx, edx и снова eax. Изменения соответствуют исходному коду программы.

Просматриваю содержимое регистров, введя `info registers` (рис. 2.12).

```

native process 3836 In: _start L17 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 2.12: Применение команды info registers

Просматриваю значение переменной msg1 по имени, msg2 - по адресу из ди-

засемблированной инструкции (рис. 2.13).

```
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) █
```

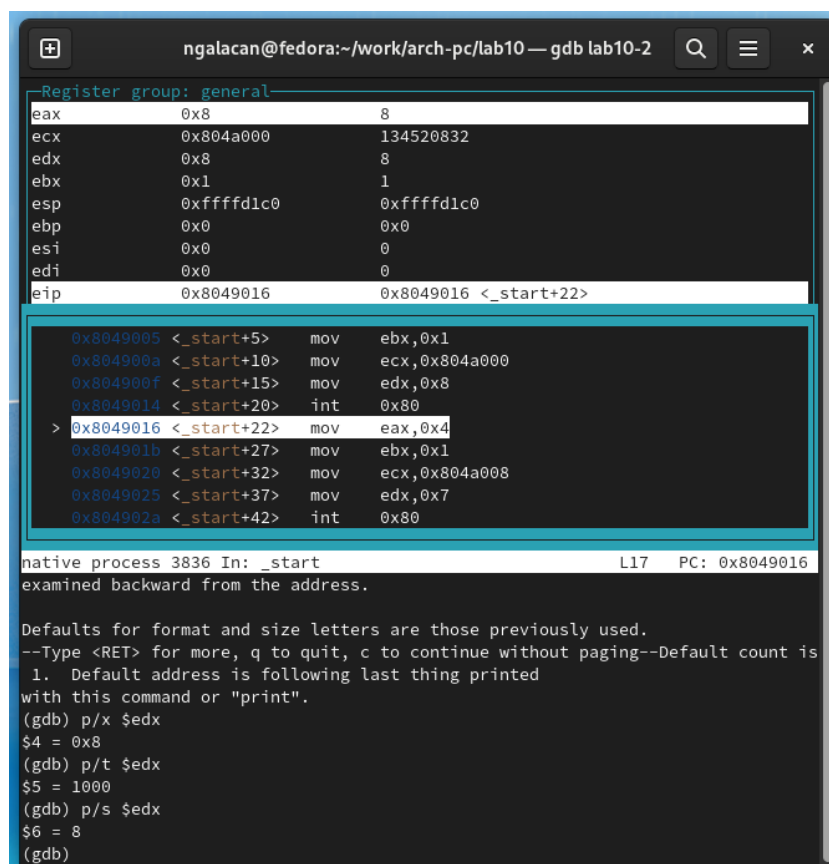
Рис. 2.13: Применение команды x для просмотра значений переменных

Изменяю первый символ переменной msg1, символ в переменной msg2 с помощью команды set (рис. 2.14).

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg2='0'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "0orld!\n\034"
(gdb) █
```

Рис. 2.14: Применение команды set

Вывожу значение регистра edx в шестнадцатеричном формате (p/x \$edx), в двоичном формате (p/t \$edx), в символьном формате (p/s \$edx) (рис. 2.15).



```
ngalacan@fedora:~/work/arch-pc/lab10 — gdb lab10-2
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>

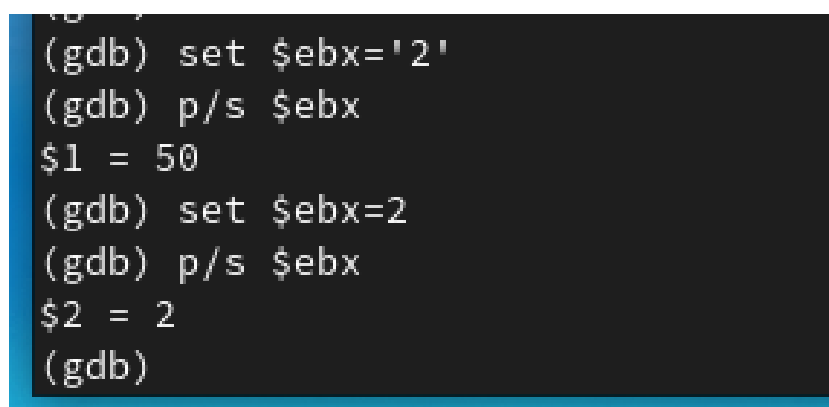
0x8049005 <_start+5>  mov     ebx,0x1
0x804900a <_start+10> mov     ecx,0x804a000
0x804900f <_start+15> mov     edx,0x8
0x8049014 <_start+20> int     0x80
> 0x8049016 <_start+22> mov     eax,0x4
0x804901b <_start+27> mov     ebx,0x1
0x8049020 <_start+32> mov     ecx,0x804a008
0x8049025 <_start+37> mov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

native process 3836 In: _start L17 PC: 0x8049016
examined backward from the address.

Defaults for format and size letters are those previously used.
--Type <RET> for more, q to quit, c to continue without paging--Default count is
1. Default address is following last thing printed
with this command or "print".
(gdb) p/x $edx
$4 = 0x8
(gdb) p/t $edx
$5 = 1000
(gdb) p/s $edx
$6 = 8
(gdb)
```

Рис. 2.15: Применение команды p в различных форматах

С помощью команды set меняю значение регистра ebx, вывожу значение с помощью p/s (рис. 2.16).



```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb)
```

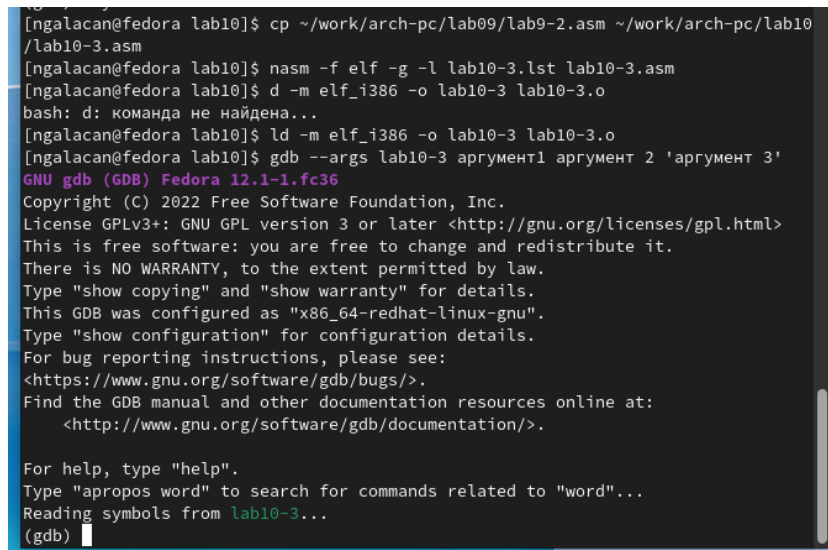
Рис. 2.16: Изменение значения регистра ebx. Просмотр

В первом случае был введен символ '2' и в качестве значения регистра был

выведен номер символа в таблице ASCII (символ 2 имеет номер 50). Во втором случае было введено и выведено само число 2.

Завершаю выполнение программы с помощью команды `continue` и выхожу из GDB с помощью команды `quit`.

Копирую файл `lab9-2.asm` с именем `lab10-3.asm` в соответствующий ЛР каталог. Создаю исполняемый файл, загружаю программу с аргументами в командной строке в отладчик, указав ключ `--args` (рис. 2.17).



```
[ngalacan@fedora lab10]$ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/
lab10-3.asm
[ngalacan@fedora lab10]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
[ngalacan@fedora lab10]$ d -m elf_i386 -o lab10-3 lab10-3.o
bash: d: команда не найдена...
[ngalacan@fedora lab10]$ ld -m elf_i386 -o lab10-3 lab10-3.o
[ngalacan@fedora lab10]$ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora 12.1-1.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. 2.17: Создание файла `lab10-3.asm`. Загрузка в отладчик

Устанавливаю точку останова перед первой инструкцией в программе, запускаю. Программа останавливается на брейкпоинте. Вывожу число аргументов (включая имя программы), которые хранятся в `esp` (рис. 2.18).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 5.
(gdb) run
Starting program: /home/ngalacan/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]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

Breakpoint 1, _start () at lab10-3.asm:5
5      pop ecx          ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd170:  0x00000005
(gdb)
```

Рис. 2.18: Установка брейкпоинта и запуск lab10-3. Вывод количества аргументов

Просматриваю остальные позиции стека с шагом измерения адреса 4 (рис. 2.19).

```
(gdb) x/x $esp
0xffffd170:  0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd32d:  "/home/ngalacan/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd357:  "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd369:  "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd37a:  "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd37c:  "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:  <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.19: Просмотр остальных позиций стека

Шаг измерения адреса равен 4, так как при добавлении в стек соответствующее значение помещается в ячейку памяти, на которую указывает регистр esp, после чего значение регистра увеличивается на 4 (т.е. “4” означает размер - 4 байта, что соответствует 32 битам).



### 3 Выполнение заданий для самостоятельной работы

1. Копирую файл lab9-sam.asm с именем lab10-sam-1.asm в соответствующий ЛР каталог. Редактирую программу, реализовав вычисление функции  $f(x)=2(x-1)$  в виде подпрограммы \_function. Преобразованная программа:

```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
fn db "Вариант 4. Функция: f(x)=2(x-1)."
```

```
jz _end
```

```
pop eax ; eax=x
```

```
call atoi
```

```
call _function
```

```
add esi,eax
```

```
loop next
```

```
_end:
```

```
mov eax,fn
```

```
call sprintf
```

```
mov eax,msg
```

```
call sprintf
```

```
mov eax,esi
```

```
call iprintLF
```

```
call quit
```

```
_function:
```

```
sub eax,1 ; eax=x-1
```

```
mov ebx,2
```

```
mul ebx ; eax=(x-1)*2
```

```
ret
```

Внутри цикла после проверки условия извлекается значение из стека, которое преобразуется в целое число. После этого вызывается функция с помощью `call`. Вычисляется значение функции, которое остается в `eax` и передается управление программе в том же месте, где был осуществлен вызов функции.

Создаю исполняемый файл и запускаю, введя такие же наборы  $x$ , как в ЛР №9. Убеждаюсь, что результат совпадает. Следовательно преобразованная программа работает правильно (рис. 3.1).

```

[ngalacan@fedora lab10]$ nasm -f elf lab10-sam-1.asm
[ngalacan@fedora lab10]$ ld -m elf_i386 -o lab10-sam-1 lab10-sam-1.o
[ngalacan@fedora lab10]$ ./lab10-sam-1 1 2 3
Вариант 4. Функция:  $f(x)=2(x-1)$ .
Результат: 6
[ngalacan@fedora lab10]$ ./lab10-sam-1 3 7 10 4 2
Вариант 4. Функция:  $f(x)=2(x-1)$ .
Результат: 42
[ngalacan@fedora lab10]$ ./lab10-sam-1 8 12
Вариант 4. Функция:  $f(x)=2(x-1)$ .
Результат: 36
[ngalacan@fedora lab10]$ ./lab10-sam-1 14 18 23 1 3 6
Вариант 4. Функция:  $f(x)=2(x-1)$ .
Результат: 118
[ngalacan@fedora lab10]$ █

```

Рис. 3.1: Запуск исполняемого файла lab10-sam-1

2. Создаю файл lab10-sam-2.asm и ввожу в него программу из листинга 10.3. Создаю исполняемый файл, запускаю. Программа выводит неверный результат (рис. 3.2).

```

[ngalacan@fedora lab10]$ touch lab10-sam-2.asm
[ngalacan@fedora lab10]$ mcedit lab10-sam-2.asm

[ngalacan@fedora lab10]$ nasm -f elf lab10-sam-2.asm
[ngalacan@fedora lab10]$ ld -m elf_i386 -o lab10-sam-2 lab10-sam-2.o
[ngalacan@fedora lab10]$ ./lab10-sam-2
Результат: 10
[ngalacan@fedora lab10]$ █

```

Рис. 3.2: Запуск исполняемого файла lab10-sam-2

Создаю исполняемый файл, оттранслировав с ключом -g для работы в GDB и загружаю в отладчик (рис. 3.3).

```

[ngalacan@fedora lab10]$ nasm -f elf -g -l lab10-2.lst lab10-sam-2.asm
[ngalacan@fedora lab10]$ ld -m elf_i386 -o lab10-sam-2 lab10-sam-2.o
[ngalacan@fedora lab10]$ gdb lab10-sam-2
GNU gdb (GDB) Fedora 12.1-1.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-sam-2...
(gdb) run
Starting program: /home/ngalacan/work/arch-pc/lab10/lab10-sam-2

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
Результат: 10
[Inferior 1 (process 5300) exited normally]
(gdb)

```

Рис. 3.3: Загрузка в отладчик lab10-sam-2

Просматриваю дисассимилированный код программы в синтаксисе Intel. Включаю режим псевдографики. Проставляю брейкпоинт на метку `_start` и запрашиваю информацию о брейкпоинтах. Запускаю программу, ввожу `si` для выполнения следующего шага. В регистр `ebx` помещается сумма значений `ebx` (3) и `eax` (2). Значение `ebx`=5. (рис. 3.4).

```
ngalacan@fedora:~/work/arch-pc/lab10 — gdb lab10-sam-2
Register group: general
eax    0x2    2
ecx    0x0    0
edx    0x0    0
ebx    0x5    5
esp    0xffffd1a0 0xffffd1a0
ebp    0x0    0x0
esi    0x0    0
edi    0x0    0

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

native process 5316 In: _start L12 PC: 0x80490f4
(gdb) break _start
Breakpoint 1 at 0x80490e8: file lab10-sam-2.asm, line 9.
(gdb) i b
Num    Type          Disp Enb Address      What
1      breakpoint    keep y  0x080490e8 lab10-sam-2.asm:9
(gdb) run
Starting program: /home/ngalacan/work/arch-pc/lab10/lab10-sam-2

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

Рис. 3.4: Установка брейкпоинта в lab10-sam-2 и просмотр. Запуск программы

Выполняю следующие шаги. В есх помещается 4, выполняется умножение. Результат умножения сохраняется в еах (т.е.  $2*4=8$ ) (рис. 3.5).

The screenshot shows a GDB terminal window with the title bar "ngalacan@fedora:~/work/arch-pc/lab10 — gdb lab10-sam-2". The window is divided into several sections:

- Register group: general**: A table showing the values of general-purpose registers.

Register	Value	Size
eax	0x8	8
ecx	0x4	4
edx	0x0	0
ebx	0x5	5
esp	0xffffd1a0	0xffffd1a0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
- Assembly code**: A list of instructions with their addresses and disassembled forms. The instruction at address 0x80490fb is highlighted.

```
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
```
- Native process**: Information about the running process.

```
native process 5316 In: _start L14 PC: 0x80490fb
```
- GDB commands and output**: A log of commands entered and their results.

```
(gdb) i b
Num  Type      Disp Enb Address  What
1    breakpoint keep y  0x080490e8 lab10-sam-2.asm:9
(gdb) run
Starting program: /home/ngalacan/work/arch-pc/lab10/lab10-sam-2

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

Рис. 3.5: Отслеживание значений регистров при вычислении выражения (1)

После этого к регистру `ebx` прибавляется 5, регистр принимает значение 10. Это значение помещается в `edi` в качестве результата. То есть, произведение, которое сохранилось в `eax` при вычислении выражение учтено не было, из-за чего выводится неверный результат (рис. 3.6).

```

ngalacan@fedora:~/work/arch-pc/lab10 — gdb lab10-sam-2
eax      0x804a000      134520832
ecx      0x4            4
edx      0x0            0
ebx      0x804a000      134520832
esp      0xffffd184     0xffffd184
ebp      0x0            0x0
esi      0x0            0
edi      0xa            10

0x8049000 <slen>      push    ebx
0x8049008 <nextchar+5> inc     eax@49003 <nextchar>
0x804900b <finished>  sub     eax,ebx
0x804900d <finished+2> pop     ebx

0x804900e <finished+3> ret
0x804900f <sprint>    push    edx
0x8049010 <sprint+1>  push    ecx
0x8049011 <sprint+2>  push    ebx

native process 5316 In: nextchar L11 PC: 0x8049008
(gdb) sNo process In: L?? PC: ??
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) c
Continuing.
Результат: 10
[Inferior 1 (process 5316) exited normally]
(gdb)

```

Рис. 3.6: Отслеживание значений регистров при вычислении выражения (2)

Выхожу из отладчика и вношу изменения в текст программы, чтобы промежуточные результаты сохранялись в регистре `eax`. Измененная программа:

```

#include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add eax,ebx

```

```
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF

call quit
```

Аналогично, как описано выше, загружаю программу в отладчик и пошагово отслеживаю значения регистров. Теперь результат сложения в скобках сохраняется в регистр `eax`. Эта сумма умножается на 4, что выше было помещено в `ecx`. Произведение сохраняется в `eax`. После этого к произведению прибавляется 5 (рис. 3.7).



```
ngalacan@fedora:~/work/arch-pc/lab10 — gdb lab10-sam-2
eax      0x19      25
eax      0x19      25
ecx      0x4       4
edx      0x0       0
ebx      0x3ffffd1a0 3ffffd1a0
esp      0xffffd1a0 0xffffd1a0
ebp      0x0       0x0
esi      0x0       05
edi      0x19      25

0x8049100 <_start+24> mov     eax,0x804a000
0x8049105 <_start+29> call   0x804900f <sprint>
0x804910a <_start+34> mov     ecx,0di
0x804910c <_start+36> call   0x8049086 <iprintLF>
0x8049111 <_start+41> call   0x80490db <quit>

>                                04a000
                                rint>

native process 5569 In: _start L17 PC: 0x8049100
Breakpoint 1: No process In: lab10-sam-2.asm:9 L?? PC: ??
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) c
Continuing.
Результат: 25
[Inferior 1 (process 5569) exited normally]
(gdb)
```

Рис. 3.7: Отладка измененной программы lab10-sam-2, вывод результата

В процессе первой отладки были выявлены ошибки, которые были исправлены. В процессе второй отладки программа была проверена на правильность работы. Теперь программа выводит верный результат.

## 4 Выводы

Приобретены навыки написания программ с использованием подпрограмм.  
Были изучены методы отладки при помощи GDB и его возможности.