

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

**Факультет физико-математических и естественных наук Кафедра
прикладной информатики и теории вероятностей**

Архипов Олег Константинович

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация подпрограмм в NASM	6
2.2	Отладка программ с помощью GDB	10
2.3	Обработка аргументов командной строки в GDB	24
3	Самостоятельная работа	27
3.1	Преобразуйте программу из лабораторной работы №8	27
3.2	Проверка программы при помощи отладчика	28
4	Выводы	41
	Список литературы	42

Список иллюстраций

2.1	Папка новой ЛР и файл lab09-1.asm	6
2.2	Код lab09-1.asm	7
2.3	Работа программы lab09-1	8
2.4	Работа измененной программы lab09-1	10
2.5	Файл lab09-2.asm	10
2.6	Код lab09-2.asm	11
2.7	Код lab09-2.asm	12
2.8	Трансляция и компоновка lab09-2.asm	12
2.9	Загрузка lab09-2 в отладчик	12
2.10	Запуск lab09-2 в оболочке GDB	13
2.11	Работа lab09-2 в оболочке GDB	13
2.12	Брейкпоинт на метке _start	13
2.13	Дизассемблированный код программы lab09-2	14
2.14	Дизассемблированный код с синтаксисом Intel	15
2.15	Режим псевдографики gdb 1	16
2.16	Режим псевдографики gdb 2	16
2.17	Точка останова break _start	17
2.18	Точка останова по адресу инструкции	17
2.19	Начало	18
2.20	si 1	18
2.21	si 2	19
2.22	si 3	19
2.23	si 4	19
2.24	si 5	20
2.25	Работа команды info registers	21
2.26	Значение переменной msg1 по имени	21
2.27	Значение переменной msg2 по адресу	21
2.28	Замена символов переменной msg1	22
2.29	Замена символов переменной msg2	22
2.30	Секция значений регистров	22
2.31	Различные форматы edx	23
2.32	Замена значения регистра ebx	24
2.33	Завершение программы и выход из отладчика	24
2.34	Файл lab09-3.asm	25
2.35	Загрузка lab09-3 в gdb с аргументами	25
2.36	Установка точки останова и запуск lab09-3	25
2.37	Количество аргументов	26

2.38	Остальные позиции стека	26
3.1	Файл sol09.asm	27
3.2	Часть кода с изменениями	28
3.3	Проверка работы программы	28
3.4	Файл для кода листинга	29
3.5	Код листинга	29
3.6	Проверка работы программы sol09-2	29
3.7	Запуск sol09-2 в gdb , точка останова и дизассемблирование . . .	30
3.8	Режим псевдографики	31
3.9	Точки останова	31
3.10	Шаг 0	32
3.11	Шаг 1	33
3.12	Шаг 2	34
3.13	Шаг 3	35
3.14	Шаг 4	36
3.15	Шаг 5	37
3.16	Шаг 6	38
3.17	Исправление кода	39
3.18	Создание исполняемого файла	39
3.19	Запуск исправленной программы	40

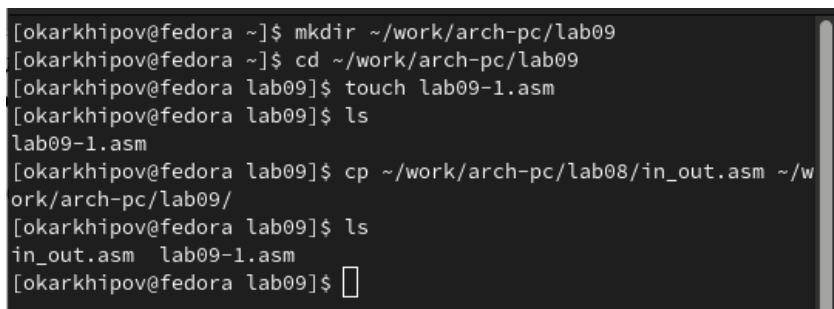
1 Цель работы

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

2 Выполнение лабораторной работы

2.1 Реализация подпрограмм в NASM

Создаю каталог для ЛР9, а также файл lab09-1.asm (рис. 2.1).



```
[okarkhipov@fedora ~]$ mkdir ~/work/arch-pc/lab09
[okarkhipov@fedora ~]$ cd ~/work/arch-pc/lab09
[okarkhipov@fedora lab09]$ touch lab09-1.asm
[okarkhipov@fedora lab09]$ ls
lab09-1.asm
[okarkhipov@fedora lab09]$ cp ~/work/arch-pc/lab08/in_out.asm ~/work/arch-pc/lab09/
[okarkhipov@fedora lab09]$ ls
in_out.asm  lab09-1.asm
[okarkhipov@fedora lab09]$
```

Рис. 2.1: Папка новой ЛР и файл lab09-1.asm

Ввожу программу вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы `_calcul` (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 iprintLF
26 call quit
27
28 ;-----
29 ; Подпрограмма вычисления
30 ; выражения "2x+7"
31 _calcul:
32 mov ebx, 2
33 mul ebx
34 add eax, 7
35 mov [res], eax
36 ret ; выход из подпрограммы

```

Рис. 2.2: Код lab09-1.asm

Создаю исполняемый файл и проверяю работу программы со значением $x=5$ (рис. 2.3).

```
[okarkhipov@fedora lab09]$ nasm -f elf lab09-1.asm
[okarkhipov@fedora lab09]$ ld -m elf_i386 lab09-1.o -o lab09-1
[okarkhipov@fedora lab09]$ ./lab09-1
Введите x: 5
2x+7=17
[okarkhipov@fedora lab09]$
```

Рис. 2.3: Работа программы lab09-1

Изменяю текст программы, добавляя 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. Помимо этого добавляю сообщение, напоминающее о возникновении сложной функции (см. листинг ниже, ср. с рис. 2.2).

Листинг программы вычисления значения сложной функции

```
%include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
result: DB 'Результат: 2*g(x)+7=',0
ad: DB 'где g(x)=3x-1',0 ; новое выражение является сложной функцией

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
mov eax, ad
call sprintLF
call quit

;-----

; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _subcalcul ; Вызов подпрограммы _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы

```

```

_subcalcul: ; подпрограмма подпрограммы
mov ebx,3   ; ebx=3
mul ebx     ; eax=eax*ebx
sub eax,1   ; eax=eax-1
ret         ; завершение _subcalcul

```

Проверяю результат (рис. 2.4).

```

[okarkhipov@fedora lab09]$ nasm -f elf lab09-1.asm
[okarkhipov@fedora lab09]$ ld -m elf_i386 lab09-1.o -o lab09-1
[okarkhipov@fedora lab09]$ ./lab09-1
Введите x: 5
Результат: 2*g(x)+7=35
где g(x)=3x-1
[okarkhipov@fedora lab09]$

```

Рис. 2.4: Работа измененной программы lab09-1

2.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm (рис. 2.5).

```

[okarkhipov@fedora lab09]$ touch lab09-2.asm
[okarkhipov@fedora lab09]$

```

Рис. 2.5: Файл lab09-2.asm

Ввожу код для вывода 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: Код lab09-2.asm

Создаю исполняемый файл lab09-2 (рис. 2.7).

```
[okarkhipov@fedora lab09]$ nasm -f elf lab09-2.asm
[okarkhipov@fedora lab09]$ ld -m elf_i386 lab09-2.o -o lab09-2
[okarkhipov@fedora lab09]$ ды
bash: ды: команда не найдена...
ls
[okarkhipov@fedora lab09]$ ls
in_out.asm  lab09-1.asm  lab09-2      lab09-2.o
lab09-1     lab09-1.o    lab09-2.asm
[okarkhipov@fedora lab09]$
```

Рис. 2.7: Код lab09-2.asm

Провожу трансляцию с ключом '-g', чтобы программу можно было отлаживать на уровне строк исходного кода (рис. 2.8).

```
[okarkhipov@fedora lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[okarkhipov@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
[okarkhipov@fedora lab09]$ ls
in_out.asm  lab09-1.asm  lab09-2      lab09-2.lst
lab09-1     lab09-1.o    lab09-2.asm  lab09-2.o
[okarkhipov@fedora lab09]$
```

Рис. 2.8: Трансляция и компоновка lab09-2.asm

Загружаю исполняемый файл в отладчик. Выводится общая информация об отладчике и полезные ссылки (рис. 2.9).

```
[okarkhipov@fedora lab09]$ gdb lab09-2
GNU gdb (GDB) Fedora Linux 13.2-6.fc38
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.htm
>
This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb)
```

Рис. 2.9: Загрузка lab09-2 в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды `r`, подтверждаю начало сессии, нажав 'y'-'ENTER' (рис. 2.10).

```
Reading symbols from lab09-2...
(gdb) r
Starting program: /home/okarkhipov/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n])
```

Рис. 2.10: Запуск lab09-2 в оболочке GDB

На экран выводится результат работы программы (рис. 2.11).

```
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
.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 7557) exited normally]
(gdb)
```

Рис. 2.11: Работа lab09-2 в оболочке GDB

Устанавливаю брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, после чего запускаю программу. Появляется краткая информация о точке останова и строке, следующей за `_start` (рис. 2.12).

```
[Inferior 1 (process 7557) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/okarkhipov/work/arch-pc/lab09/lab09-2

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

Рис. 2.12: Брейкпоинт на метке `_start`

Ввожу команду `disassemble _start`, чтобы посмотреть дизассемблированный код программы (рис. 2.13).

```

(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. 2.13: Дизассемблированный код программы lab09-2

Переключаюсь на отображение команд с Intel'овским синтаксисом:

```

(gdb) set disassembly-flavor intel
(gdb) disassemble _start

```

Можно сравнить рис. 2.13 и рис. 2.14. В последней колонке режима АТТ сначала указан адрес второго операнда, а затем со знаком '%' - первый. Синтаксис Intel больше похож на исходный код, как он вводится в текстовый редактор: сохранен порядок адресов операндов и нет '%'.

```

(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. 2.14: Дизассемблированный код с синтаксисом Intel

Ввожу команды для перехода в режим псевдографики. Как видим, значения регистров недоступны (рис. 2.15), так что снова ввожу команду `run`, после чего значения регистров появляются в верхней секции (рис. 2.16).

```

(gdb) layout asm
(gdb) layout regs

```

```

[ 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
      0x8049020 <_start+32> mov    ecx,0x804a008
      0x8049025 <_start+37> mov    edx,0x7
      0x804902a <_start+42> int     0x80

native process 7649 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 2.15: Режим псевдографики gdb 1

```

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]

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

native process 12971 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/okarkhipov/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9

```

Рис. 2.16: Режим псевдографики gdb 2

Проверяю наличие введенной ранее точки останова (рис. 2.17).

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

Рис. 2.17: Точка останова break_start

Создаю новую точку останова по адресу предпоследней инструкции (mov ebx,0x0) и проверяю, какие точки останова созданы (рис. 2.18).

```
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],al

native process 12971 In: _start
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000  lab09-2.asm:9
          breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000  lab09-2.asm:9
          breakpoint already hit 1 time
2        breakpoint     keep y   0x08049031  lab09-2.asm:20
(gdb) █
```

Рис. 2.18: Точка останова по адресу инструкции

С помощью команды stepi (или si) выполняю 5 инструкций (рис. 2.19-2.24). Как видим, на каждом шаге измененные регистры выделены белым: 2 раза eax, ebx, 3 раза eip, ecx, edx.

```

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0

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],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 12971 In: _start L9 PC: 0x8049000
(gdb) 

```

Рис. 2.19: Начало

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1b0 0xffffd1b0
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
es       0x2b     43
fs       0x0      0
gs       0x0      0

```

Рис. 2.20: si 1

Register group: general		
eax	0x4	4
ecx	0x0	0
edx	0x0	0
ebx	0x1	1
esp	0xffffd1b0	0xffffd1b0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x804900a	0x804900a <_start+10>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x0	0
gs	0x0	0

Рис. 2.21: si 2

Register group: general		
eax	0x4	4
ecx	0x804a000	134520832
edx	0x0	0
ebx	0x1	1
esp	0xffffd1b0	0xffffd1b0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x804900f	0x804900f <_start+15>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x0	0
gs	0x0	0

Рис. 2.22: si 3

Register group: general		
eax	0x4	4
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xffffd1b0	0xffffd1b0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049014	0x8049014 <_start+20>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x0	0
gs	0x0	0

Рис. 2.23: si 4

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0
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

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],al
0x804903a          add     BYTE PTR [eax],al
0x804903c          add     BYTE PTR [eax],al

native process 12971 In: _start L14 PC: 0x8049016
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) 

```

Рис. 2.24: si 5

Далее ввожу команду `i r`, чтобы просмотреть содержимое регистров другим способом (рис. 2.25).

```

eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0
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
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 2.25: Работа команды info registers

Смотрю значение переменной msg1 по имени (рис. 2.26).

```

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

```

Рис. 2.26: Значение переменной msg1 по имени

Смотрю значение переменной msg2 по адресу, определяя его по дизассемблированной инструкции mov ecx,msg2 (рис. 2.27).

```

> 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

native process 13952 In: _start L14 PC: 0x8049016
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--RETeS
0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) 

```

Рис. 2.27: Значение переменной msg2 по адресу

Заменяю первый и второй символы переменной msg1 (рис. 2.28).

```
(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hhlllo, "
```

Рис. 2.28: Замена символов переменной msg1

Заменяю третий символ переменной msg2 ('r' на 'w') (рис. 2.29).

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

Рис. 2.29: Замена символов переменной msg2

Вывожу в различных форматах значение регистра edx (p/x и p/a - символьный, p/t - двоичный, p/s - шестнадцатиричный) (рис. 2.30-2.31).

Register group: general		
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xffffd1b0	0xffffd1b0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 <_start+22>
eflags	0x202	[IF]
cs	0x23	35

Рис. 2.30: Секция значений регистров

```
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/a $edx
$3 = 0x8
(gdb) p/s $edx
$4 = 8
(gdb) 
```

Рис. 2.31: Различные форматы edx

С помощью команды `set` изменяю значение регистра `ebx` (рис. 2.32). Результаты различны из-за того, что в первом случае значение `ebx` было заменено на символ '2', код которого в ASCII равен 50, во втором случае значение `ebx` заменяется на код.

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

Рис. 2.32: Замена значения регистра ebx

Завершаю выполнение программы при помощи `continue` и выхожу при помощи `q` (рис. 2.33).

```
(gdb) continue
Continuing.
wow!d!
[Inferior 1 (process 13952) exited normally]
(gdb) q
```

Рис. 2.33: Завершение программы и выход из отладчика

2.3 Обработка аргументов командной строки в GDB

Копирую файл `lab8-2.asm` в `lab09` с именем `lab09-3.asm`, создаю исполняемый файл (рис. 2.34).


```
[okarkhipov@fedora lab09]$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
[okarkhipov@fedora lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[okarkhipov@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[okarkhipov@fedora lab09]$
```

Рис. 2.34: Файл lab09-3.asm

Используя ключ `--args`, загружаю программу в gdb с аргументами (рис. 2.35).

```
[okarkhipov@fedora lab09]$ gdb --args lab09-3 4 2 '7'
```

Рис. 2.35: Загрузка lab09-3 в gdb с аргументами

Для исследования расположения аргументов командной строки в стеке после запуска программы с помощью gdb устанавливаю точку останова перед первой инструкцией в программе и запускаю ее (рис. 2.36).

```
(gdb) b _start
```

```
(gdb) run
```

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/okarkhipov/work/arch-pc/lab09/lab09-3 4 2 7

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 2.36: Установка точки останова и запуск lab09-3

Вывожу адрес вершины стека, где располагается число равное количеству аргументов командной строки (включая имя программы), их 4 (рис. 2.37).

```
(gdb) x/x $esp
0xffffd190:      0x00000004
(gdb) █
```

Рис. 2.37: Количество аргументов

Просматриваю остальные позиции стека – по адресу [esp+4] располагается адрес в памяти, где находится имя программы, по адресу [esp+8] хранится адрес первого аргумента, по адресу [esp+12] – второго, по адресу [esp+16] - третьего (рис. 2.38). Шаг изменения равен 4, т.к. данные аргументы смещены относительно вершины стека на 4, 8 и т.д. байта.

```
(gdb) x/s *(void**)(esp + 4)
0xffffd34e:      "/home/okarkhipov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd37a:      "4"
(gdb) x/s *(void**)(esp + 12)
0xffffd37c:      "2"
(gdb) x/s *(void**)(esp + 15)
0xffd37eff:      ""
(gdb) x/s *(void**)(esp + 16)
0xffffd37e:      "7"
(gdb) x/s *(void**)(esp + 20)
0x0:      <error: Cannot access memory at address 0x0>
(gdb) █
```

Рис. 2.38: Остальные позиции стека

3 Самостоятельная работа

3.1 Преобразуйте программу из лабораторной работы №8

Копирую файл самостоятельной работы из ЛР8 с новым именем (рис. 3.1).

```
[okarkhipov@fedora lab09]$ cp ~/work/arch-pc/lab08/sol8.asm ~/work/arch-pc/lab09/sol09.asm
[okarkhipov@fedora lab09]$ ls
in_out.asm  lab09-1.o  lab09-2.lst  lab09-3.asm  sol09.asm
lab09-1     lab09-2   lab09-2.o    lab09-3.lst
lab09-1.asm lab09-2.asm lab09-3      lab09-3.o
[okarkhipov@fedora lab09]$
```

Рис. 3.1: Файл sol09.asm

Редактирую код так, чтобы вычисление значений $f(x)$ стало подпрограммой (рис. 3.2).

```

next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end    ; если аргументов нет выходим из цикла
           ; (переход на метку `_end`)
call _f    ; вызов подпрограммы вычисления f(x)
add esi, eax ; прибавляем к промежуточной сумме
loop next  ; переход к обработке следующего аргумента
_end:
mov eax, f  ; вывод сообщения "Функция: f(x)=2(x-1)"
call printf
mov eax,msg ; вывод сообщения "Результат: "
call printf
mov eax, esi ; записываем сумму в регистр `eax`
call printf ; печать результата
call quit  ; завершение программы

_f:
pop eax    ; иначе извлекаем следующий аргумент из стека
call atoi  ; преобразуем символ в число
sub eax, 1 ; eax=eax-1
mov edi, 2 ; edi=2
mul edi    ; eax=eax*edi
push eax
ret

```

Рис. 3.2: Часть кода с изменениями

Проверка корректности работы измененного кода (рис. 3.3).

```

[okarkhipov@fedora lab09]$ nasm -f elf sol09.asm
[okarkhipov@fedora lab09]$ ld -m elf_i386 sol09.o -o main
[okarkhipov@fedora lab09]$ ./main 14 2 73 9
Функция: f(x)=2(x-1)
Результат: 188

```

Рис. 3.3: Проверка работы программы

3.2 Проверка программы при помощи отладчика

Создаю файл sol09-2.asm и ввожу код из листинга для вычисления выражения $(3 + 2) * 4 + 5$ (рис. 3.4-3.5).

```
[okarkhipov@fedora lab09]$ touch sol09-2.asm  
[okarkhipov@fedora lab09]$
```

Рис. 3.4: Файл для кода листинга

```
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 ebx,eax  
11 mov ecx,4  
12 mul ecx  
13 add ebx,5  
14 mov edi,ebx  
15 ; ---- Вывод результата на экран  
16 mov eax,div  
17 call sprint  
18 mov eax,edi  
19 call iprintLF  
20 call quit
```

Рис. 3.5: Код листинга

Проверка показывает ошибочный ответ (рис. 3.6).

```
[okarkhipov@fedora lab09]$ nasm -f elf -g -l sol09-2.lst sol09-2.asm  
[okarkhipov@fedora lab09]$ ld -m elf_i386 -o sol09-2 sol09-2.o  
[okarkhipov@fedora lab09]$ ./sol09-2  
Результат: 10  
[okarkhipov@fedora lab09]$
```

Рис. 3.6: Проверка работы программы sol09-2

Запускаю sol09-2 в gdb , ставлю точку останова на _start , т.к. ошибка, очевидно, в теле программы и дизассемблирую ее при помощи синтаксиса Intel (рис. 3.7).

```
[okarkhipov@fedora lab09]$ gdb sol09-2
GNU gdb (GDB) Fedora Linux 13.2-6.fc38
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from sol09-2...
(gdb) run
Starting program: /home/okarkhipov/work/arch-pc/lab09/sol09-2

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
.
Результат: 10
[Inferior 1 (process 19424) exited normally]
(gdb) break _start
Breakpoint 1 at 0x80490e8: file sol09-2.asm, line 8.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
```

Рис. 3.7: Запуск sol09-2 в gdb , точка останова и дизассемблирование

Включаю режим псевдографики (рис. 3.8).

```

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490e8 0x80490e8 <_start>
eflags   0x202    [ IF ]
cs       0x23     35

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 <iprintfLF>
0x8049111 <_start+41> call   0x80490db <quit>

native process 19502 In: _start L8 PC: 0x80490e8
(gdb) layout regs
(gdb) run
Starting program: /home/okarkhipov/work/arch-pc/lab09/sol09-2

Breakpoint 1, _start () at sol09-2.asm:8

```

Рис. 3.8: Режим псевдографики

Устанавливаю точки останова после каждого действия (рис. 3.9). Буду проверять значения регистров `eax`, `ebx`, `ecx` на каждом шаге, т.к. они участвуют в вычислении значения выражения.

```

(gdb) break *0x80490ed
Breakpoint 2 at 0x80490ed: file sol09-2.asm, line 9.
(gdb) b *0x80490f2
Breakpoint 3 at 0x80490f2: file sol09-2.asm, line 10.
(gdb) b *0x80490f4
Breakpoint 4 at 0x80490f4: file sol09-2.asm, line 11.
(gdb) b *0x80490f9
Breakpoint 5 at 0x80490f9: file sol09-2.asm, line 12.
(gdb) b *0x80490fb
Breakpoint 6 at 0x80490fb: file sol09-2.asm, line 13.
(gdb) b *0x80490fe
Breakpoint 7 at 0x80490fe: file sol09-2.asm, line 14.
(gdb) 

```

Рис. 3.9: Точки останова

Запустив программу, и введя команды

```
(gdb) p/d $eax
```

```
$1=0
```

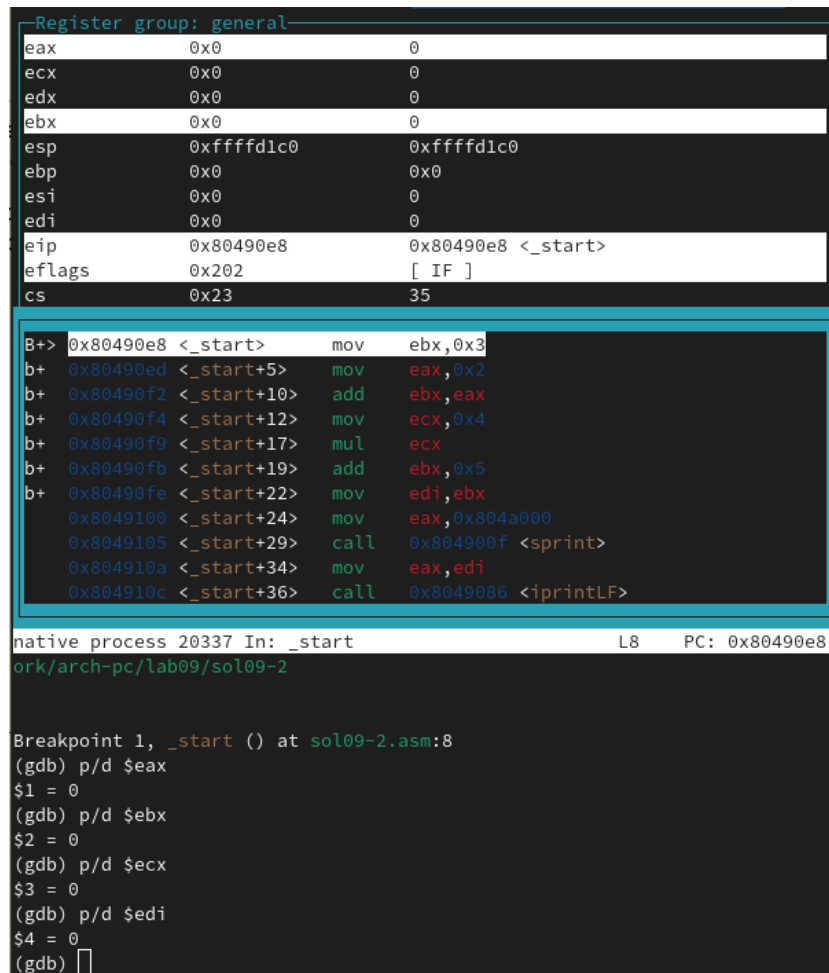
```
(gdb) p/d $ebx
```

```
$2=0
```

```
(gdb) p/d $ecx
```

```
$3=0
```

где d отвечает за отображение результатов в десятичном формате, получаю пока значения во всех 3-х регистрах, равные нулю (рис. 3.10).



The screenshot shows a debugger window with two main panes. The top pane, titled 'Register group: general', displays the values of various registers. The bottom pane shows the assembly code being executed, with the current instruction highlighted. Below the assembly pane, the status bar indicates the current instruction pointer (PC) and the process name. At the bottom, the breakpoint list shows a breakpoint set at the start of the program.

Register	Value	Comment
eax	0x0	0
ecx	0x0	0
edx	0x0	0
ebx	0x0	0
esp	0xffffd1c0	0xffffd1c0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80490e8	0x80490e8 <_start>
eflags	0x202	[IF]
cs	0x23	35

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

native process 20337 In: _start L8 PC: 0x80490e8
ork/arch-pc/lab09/sol09-2

Breakpoint 1, _start () at sol09-2.asm:8
(gdb) p/d \$eax
\$1 = 0
(gdb) p/d \$ebx
\$2 = 0
(gdb) p/d \$ecx
\$3 = 0
(gdb) p/d \$edi
\$4 = 0
(gdb) □

Рис. 3.10: Шаг 0

Далее проматываю программу до следующей точки останова (на 1 строку) при помощи с . В верхней секции вижу, что из интересующих меня изменился только регистр ebx , он стал равен 3 (рис. 3.11).

```

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490ed 0x80490ed <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35

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

native process 20337 In: _start L9 PC: 0x80490ed
(gdb) p/d $ebx
$2 = 0
(gdb) p/d $ecx
$3 = 0
(gdb) p/d $edi
$4 = 0
(gdb) c
Continuing.

Breakpoint 2, _start () at sol09-2.asm:9
(gdb) p/d $ebx
$5 = 3
(gdb) 

```

Рис. 3.11: Шаг 1

Так же проматываю далее. Изменился eax , ему присвоено значение 2 (рис. 3.12).

```

Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f2 0x80490f2 <_start+10>
eflags   0x202    [ IF ]
cs       0x23     35

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

native process 20337 In: _start L10 PC: 0x80490f2
(gdb) c
Continuing.

Breakpoint 2, _start () at sol09-2.asm:9
(gdb) p/d $ebx
$5 = 3
(gdb) c
Continuing.

Breakpoint 3, _start () at sol09-2.asm:10
(gdb) p/d $eax
$6 = 2
(gdb) 

```

Рис. 3.12: Шаг 2

Теперь эти два числа нужно сложить, за что отвечает следующая строка. Операция выполнена верно (получено число 5), и результат записан в регистр ebx (рис. 3.13).

```
Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x5      5
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f4 0x80490f4 <_start+12>
eflags   0x206    [ PF IF ]
cs       0x23     35

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

native process 20337 In: _start L11 PC: 0x80490f4

Breakpoint 3, _start () at sol09-2.asm:10
(gdb) p/d $eax
$6 = 2
(gdb) c
Undefined command: ". Try "help".
(gdb) c
Continuing.

Breakpoint 4, _start () at sol09-2.asm:11
(gdb) p/d $ebx
$7 = 5
(gdb) □
```

Рис. 3.13: Шаг 3

Далее в регистр ecx было записано число 4 (рис. 3.14).

```

Register group: general
eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f9 0x80490f9 <_start+17>
eflags   0x206    [ PF IF ]
cs       0x23     35

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

native process 20337 In: _start L12 PC: 0x80490f9
(gdb) c
Continuing.

Breakpoint 4, _start () at sol09-2.asm:11
(gdb) p/d $ebx
$7 = 5
(gdb) c
Continuing.

Breakpoint 5, _start () at sol09-2.asm:12
(gdb) p/d $ecx
$8 = 4
(gdb) 

```

Рис. 3.14: Шаг 4

Перехожу к следующему действию. Теперь в регистре `eax` стоит число 8 (рис. 3.15). Что является результатом операции `'eax=eax*ecx=24'`, это неверно, т.к. необходима операция `'eax=ebx*ecx=54=20'` из-за того что результат первого действия был записан в регистр `ebx`, однако когда размер операндов 4 байта один из сомножителей всегда берется из регистра `eax` и записывается в него же. Итак, необходимо записать результат первого сложения в регистр `eax`.

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490fb 0x80490fb <_start+19>
eflags   0x202    [ IF ]
cs       0x23     35

B+ 0x80490e8 <_start>    mov     ebx,0x3
B+ 0x80490ed <_start+5>  mov     eax,0x2
B+ 0x80490f2 <_start+10> add     ebx,eax
B+ 0x80490f4 <_start+12> mov     ecx,0x4
B+ 0x80490f0 <_start+17> mul     ecx
B+ 0x80490fb <_start+19> add     ebx,0x5
b+ 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 <iprintf>

native process 20337 In: _start L13 PC: 0x80490fb
(gdb) c
Continuing.

Breakpoint 5, _start () at sol09-2.asm:12
(gdb) p/d $ecx
$8 = 4
(gdb) c
Continuing.

Breakpoint 6, _start () at sol09-2.asm:13
(gdb) p/d $eax
$9 = 8
(gdb) 

```

Рис. 3.15: Шаг 5

На всякий случай продолжу проверку. После перехода к следующему шагу получаю 'ebx=ebx+5=10'. Это итоговый ответ, и он неверный, т.к. в регистре ebx по сказанному выше не учитывается результат умножения (рис. 3.16).

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490fe 0x80490fe <_start+22>
eflags   0x206    [ PF IF ]
cs       0x23     35

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

native process 20337 In: _start L14 PC: 0x80490fe
(gdb) c
Continuing.

Breakpoint 6, _start () at sol09-2.asm:13
(gdb) p/d $eax
$9 = 8
(gdb) c
Continuing.

Breakpoint 7, _start () at sol09-2.asm:14
(gdb) p/d $ebx
$10 = 10
(gdb) 

```

Рис. 3.16: Шаг 6

Итак, чтобы программа работала корректно, нужно записать результат первого и последнего действий в регистр 'eax' вместо 'ebx', а также поменять первое слогаемое в последнем действии на eax. Кроме того, т.к. для записи итогового значения используется регистр eax, то в edi нужно поместить значение eax

Выхожу из отладчика и исправляю код (рис. 3.17).

```

GNU nano 7.2 /home/okarkhipov/work/arch-pc
%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

```

Рис. 3.17: Исправление кода

Сохраняю изменения, выхожу из редактора, создаю исполняемый файл и запускаю его (рис. 3.18-3.19).

```

[okarkhipov@fedora lab09]$ nasm -f elf sol09-2.asm
[okarkhipov@fedora lab09]$ ld -m elf_i386 sol09-2.o -o sol09-2
[okarkhipov@fedora lab09]$ 

```

Рис. 3.18: Создание исполняемого файла

```
[okarkhipov@fedora lab09]$ ./sol09-2  
Результат: 25  
[okarkhipov@fedora lab09]$
```

Рис. 3.19: Запуск исправленной программы

Получаю верно работающую программу. Хууух

4 Выводы

Освоены навыки написания программ при помощи подпрограмм и методы отладки при помощи gdb.

Список литературы