

Лабораторная работа №9

Понятие подпрограммы.Отладчик GDB.

Казначеев Сергей Ильич

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
3	Выполнение заданий для самостоятельной работы	24
4	Выводы	32

Список иллюстраций

2.1 01	7
2.2 02	7
2.3 03	8
2.4 04	9
2.5 05	9
2.6 06	10
2.7 07	10
2.8 08	11
2.9 09	11
2.10 10	12
2.11 11	12
2.12 12	13
2.13 13	13
2.14 14	14
2.15 15	14
2.16 16	15
2.17 17	16
2.18 18	16
2.19 19	17
2.20 20	17
2.21 21	18
2.22 22	19
2.23 23	19
2.24 24	19
2.25 25	19
2.26 26	20
2.27 27	20
2.28 28	20
2.29 29	21
2.30 30	21
2.31 31	21
2.32 32	21
2.33 33	22
2.34 34	22
2.35 35	22
2.36 36	23

3.1	37	24
3.2	38	25
3.3	39	26
3.4	40	26
3.5	41	27
3.6	42	27
3.7	43	28
3.8	44	28
3.9	45	28
3.10	46	29
3.11	47	29
3.12	48	30
3.13	49	30

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

Для начала создадим папку и файл lab9-1.asm

```
kava@fedora:~$ mkdir ~/work/arch-pc/lab09
kava@fedora:~$ cd ~/work/arch-pc/lab09
kava@fedora:~/work/arch-pc/lab09$ touch lab09-1.asm
kava@fedora:~/work/arch-pc/lab09$
```

Рис. 2.1: 01

Далее, запустим Midnight commander и копируем файл in_out.asm.

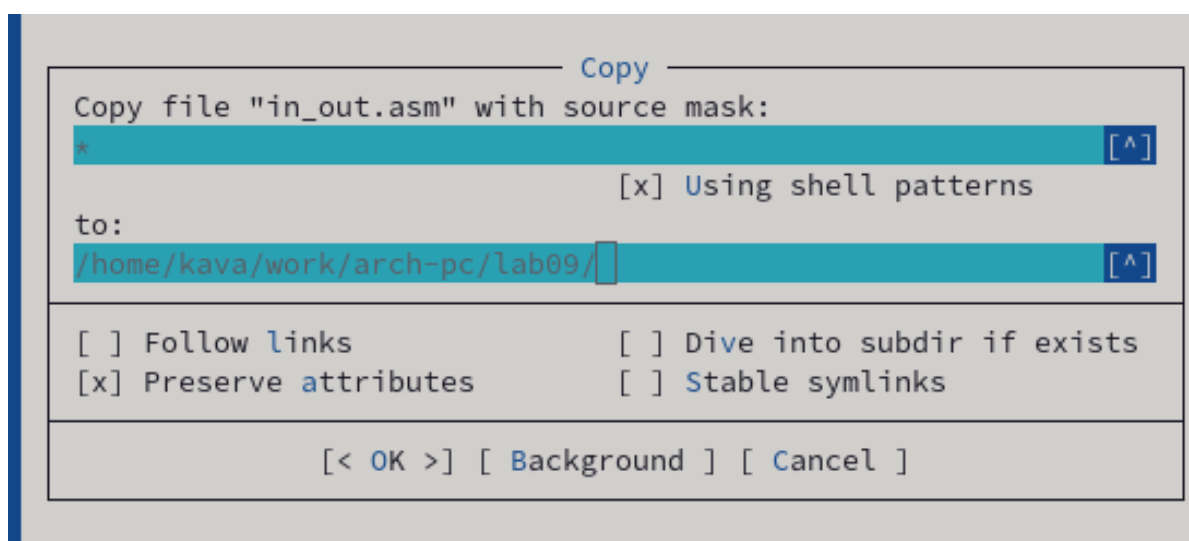


Рис. 2.2: 02

Вставляем в файл lab9-1.asm код из листинга 9.1

```

GNU nano 7.2 /home/kava/work/arch-pc/lab09/l
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
;-----
; Подпрограмма вычисления

```

Рис. 2.3: 03

Собираем программу и проверяем её на корректность работы


```

kava@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
kava@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
kava@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
kava@fedora:~/work/arch-pc/lab09$ █

```

Рис. 2.4: 04

Теперь изменим файл так, чтобы внутри подпрограммы была ещё одна подпрограмма, вычисляющая значение $g(x)$ и чтобы она передавала значение в первую подпрограмму, которая бы уже вычислила значение $f(g(x))$

```

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

ret ; выход из подпрограммы
█

```

Рис. 2.5: 05

Собираем и проверяем

```
kava@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
kava@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
kava@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 1
f(g(x))=11
kava@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
f(g(x))=23
kava@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 2
f(g(x))=17
```

Рис. 2.6: 06

Создаем второй файл

```
kava@fedora:~/work/arch-pc/lab09$ touch lab09-2.asm
kava@fedora:~/work/arch-pc/lab09$
```

Рис. 2.7: 07

И вставляем код из листинга 9.2

```

GNU nano 7.2                               /home/kava/work/ar
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. 2.8: 08

Соберем программу следующим образом

```

kava@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
kava@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
kava@fedora:~/work/arch-pc/lab09$ 

```

Рис. 2.9: 09

Теперь загружаем ее в gdb

```
kava@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 15.2-2.fc40
Copyright (C) 2024 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 lab09-2...
(gdb) █
```

Рис. 2.10: 10

Запускаем ее с помощью команды run

```
(gdb) run
Starting program: /home/kava/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 7237) exited normally]
(gdb) █
```

Рис. 2.11: 11

Создадим брейкпоинт на метке _start с помощью команды break

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/kava/work/arch-pc/lab09/lab09-2

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

```

Рис. 2.12: 12

С помощью команды disassemble дизассемблируем её

```

(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: 13

Переключаем вывода синтекса на intel

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

Рис. 2.14: 14

Повторяем команду disassemble

```
(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. 2.15: 15

Включаем графическое отображение кода

```
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
0x8049031 <_start+49>     mov    ebx,0x0
0x8049036 <_start+54>     int     0x80

native process 7362 (asm) In: _start
(gdb) █
```

Рис. 2.16: 16

Теперь включаем графическое отображение значений регистров

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd080 0xffffd080
ebp      0x0      0x0

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

native process 7362 (asm) In: _start
(gdb) layout regs
(gdb) □
```

Рис. 2.17: 17

Выводим всю информацию о всех брейкпоинтах

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

Рис. 2.18: 18

Создаем брейкпоинт по адресу


```
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) 
```

Рис. 2.19: 19

Выводим информацию

```
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. 2.20: 20

Теперь 5 раз выполняем команду `si` для построчного выполнения кода

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd080 0xffffd080
ebp      0x0      0x0
esi      0x0      0

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

native process 7362 (asm) In: _start
1      breakpoint      keep y    0x08049000 lab09-2.asm:9
      breakpoint already hit 1 time
2      breakpoint      keep y    0x08049031 lab09-2.asm:20
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) 
```

Рис. 2.21: 21

Как видим, поменялись значения регистров `eax`, `ecx`, `edx` и `ebx`. Теперь выведем информацию о значениях регистров

```

native process 7362 (asm) In: _start L14
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd080 0xffffd080
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 2.22: 22

Выводим значения переменной по имени.

```

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

```

Рис. 2.23: 23

Теперь по адресу

```

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

```

Рис. 2.24: 24

Теперь меняем первый символ

```

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

```

Рис. 2.25: 25

Меняем второй символ

```

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

```

Рис. 2.26: 26

Меняем несколько символов второй переменной

```

(gdb) x/1sb &msg2
0x804a008 <msg2>:      "Lor d!\n\034"
(gdb) 

```

Рис. 2.27: 27

Выводим значения регистра в строковом, двоичном и шестнадцатиричном виде

```

(gdb) print /s $edx
$1 = 8
(gdb) print /t $edx
$2 = 1000
(gdb) print /x $edx
$3 = 0x8
(gdb) 

```

Рис. 2.28: 28

Пробуем изменить значения регистра

```

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

```

Рис. 2.29: 29

Мы увидим что в регистр записались разные значения, это связано с тем, что в одном случае мы записываем в него число, а в другом случае строчку. Завершаем программу с помощью команды continue и выйдем.

```

(gdb) continue
Continuing.
Lor d!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb) 

```

Рис. 2.30: 30

```

End of assembler dump.
(gdb) layout asm
kava0fedora:~/work/arch-pc/lab09$ 

```

Рис. 2.31: 31

Копируем файл

```

ava@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
ava@fedora:~/work/arch-pc/lab09$ 

```

Рис. 2.32: 32

Соберём его и вгрузим в gdb

```

kava@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
kava@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
kava@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Fedora Linux) 15.2-2.fc40
Copyright (C) 2024 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 lab09-3...
(gdb) 

```

Рис. 2.33: 33

Создадим брейкпоинт и запустим программу

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/kava/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

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

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

```

Рис. 2.34: 34

Выведем значение регистра есп, где хранятся данные о стеке

```

(gdb) x/x $esp
0xffffd030:      0x00000005
(gdb) 

```

Рис. 2.35: 35

Выведем значения всех элементов стека

```
(gdb) x/x $esp
0xffffd030:      0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffd1fa:      "/home/kava/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd220:      "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd232:      "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd243:      "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd245:      "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:      <error: Cannot access memory at address 0x0>
(gdb) □
```

Рис. 2.36: 36

Как видим, для вывода каждого элемента стека нам нужно менять значение адреса с шагом 4. Это связано с тем, что именно с шагом 4 располагаются данные в стеке.

3 Выполнение заданий для самостоятельной работы

Копируем файл первого задания прошлой самостоятельной работы

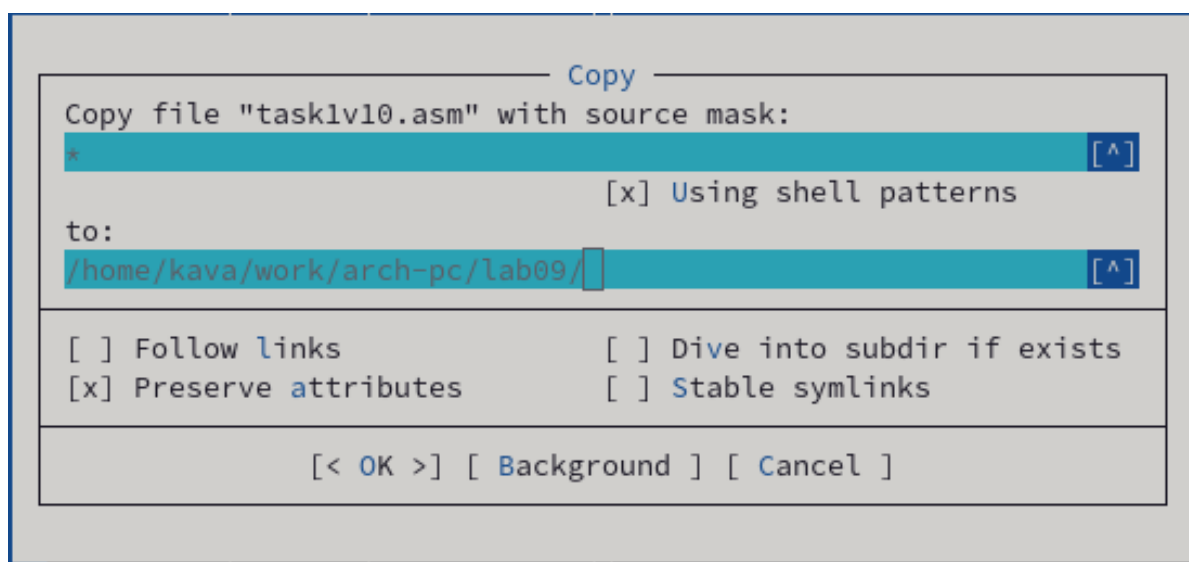


Рис. 3.1: 37

Переписываем так, чтобы он использовался для авчисления выражения под-
программы


```

include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg2 db "Функция: f(x)=5(2+x)",0
SECTION .text
global _start
_start:
    pop ecx          ; Извлекаем из стека в `ecx` количество аргументов (первое значение)
    pop edx          ; Извлекаем из стека в `edx` имя программы (второе значение)
    sub ecx, 1       ; Уменьшаем `ecx` на 1 (количество аргументов без названия программы)
    mov esi, 0       ; Используем `esi` для хранения промежуточных сумм
next:
    cmp ecx, 0       ; Проверяем, есть ли еще аргументы
    jz _end          ; Если аргументов нет, выходим из цикла (переход на метку `_end`)
    pop eax          ; Извлекаем следующий аргумент из стека
    call atoi        ; Преобразуем символ в число
    ; Применяем новую формулу f(x) = 5(2 + x)
    call _calcul
    add esi, eax      ; Добавляем к промежуточной сумме результат: `esi = esi + eax`
    loop next        ; Переход к обработке следующего аргумента
_end:
    mov eax, msg2     ; Вывод сообщения "Функция: f(x)=5(2+x)"
    call sprintf
    mov eax, msg      ; Вывод сообщения "Результат: "
    call sprintf
    mov eax, esi      ; Записываем сумму в регистр `eax`
    call iprintf
    call quit        ; Завершение программы
_calcul:
    add eax, 2
    mov ebx, 5
    mul ebx
    ret

```

Рис. 3.2: 38

Собираем и проверяем на корректность выполнения.

```
kava@fedora:~/work/arch-pc/lab09$ nasm -f elf task1v10.asm
kava@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o task1v10 task1v10.o
kava@fedora:~/work/arch-pc/lab09$ ./task1v10 1 2 3 4
Функция:  $f(x)=5(2+x)$ 
Результат: 90
kava@fedora:~/work/arch-pc/lab09$ ./task1v10 1
Функция:  $f(x)=5(2+x)$ 
Результат: 15
kava@fedora:~/work/arch-pc/lab09$ ./task1v10 1 5
Функция:  $f(x)=5(2+x)$ 
Результат: 50
```

Рис. 3.3: 39

Создадим файл второго задания самостоятельной работы

```
kava@fedora:~/work/arch-pc/lab09$ touch task2.asm
kava@fedora:~/work/arch-pc/lab09$
```

Рис. 3.4: 40

Далее вставляем код из листинга 9.3

```

GNU nano 7.2 /
#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 ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 3.5: 41

Собираем и запускаем

```

kava@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l task2.lst task2.asm
kava@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o task2 task2.o
kava@fedora:~/work/arch-pc/lab09$ ./task2
Результат: 10

```

Рис. 3.6: 42

Как видим, код считает значение выражения неправильно. Загрузим его в gdb.

```

kava@fedora:~/work/arch-pc/lab09$ gdb task2
GNU gdb (Fedora Linux) 15.2-2.fc40
Copyright (C) 2024 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 task2...
(gdb) 

```

Рис. 3.7: 43

Включением графическое отображение значений регистров и отображение графического отображения кода.

```

exec No process (asm) In:
(gdb) layout regs
(gdb) 

```

Рис. 3.8: 44

Устанавливаем брейкпоинт на `_start`

```

(gdb) break _start
Breakpoint 1 at 0x80490e8: file task2.asm, line 8.
(gdb) 

```

Рис. 3.9: 45

Запускаем и начинаем построчно выполнять код

```

(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kava/work/arch-pc/lab09/task2

Breakpoint 1, _start () at task2.asm:8
(gdb) █

```

Рис. 3.10: 46

```

Register group: general
eax      0x8      8      ecx      0x4      4
edx      0x0      0      ebx      0xa      10
esp      0xffffd070 0xffffd070  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x80490fe 0x80490fe <_start+22>  eflags    0x10206    [ PF IF RF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       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
0x8049105 <_start+29>   call   0x804900f <sprint>
0x804910a <_start+34>   mov     eax,edi
0x804910c <_start+36>   call   0x8049086 <iprintf>
0x8049111 <_start+41>   call   0x80490db <quit>

native process 15159 (asm) In: _start      L14    PC: 0x80490fe
Breakpoint 1, _start () at task2.asm:8
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kava/work/arch-pc/lab09/task2

Breakpoint 1, _start () at task2.asm:8
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) █

```

Рис. 3.11: 47

Как видим, мы должны были умножить значение регистра ebx, но умножили регистр eax. Нам необходимо все результаты хранить в регистре eax. Изменим код

```

GNU nano 7.2
%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. 3.12: 48

И проверяем на корректность выполнения.

```

kava@fedora:~/work/arch-pc/lab09$ mc
kava@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l task2.lst task2.asm
kava@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o task2 task2.o
kava@fedora:~/work/arch-pc/lab09$ ./task2
Результат: 25
kava@fedora:~/work/arch-pc/lab09$ 

```

Рис. 3.13: 49

Как видим, теперь код работает корректно

4 Выводы

После выполнения лабораторной работы. Я приобрел навыки программ с использованием подпрограмм и познакомился с методами отладки при помощи GDB и его основными возможностями