

Лабораторная работа №9

Понятие подпрограммы. Отладчик GDB

Павличенко Родион Андреевич

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
3	Выполнение задания для самостоятельной работы	20
4	Выводы	29

Список иллюстраций

2.1	Создание рабочей директории и файла lab9-1.asm	7
2.2	Запуск Midnight commander	7
2.3	Копирование файла in_out.asm в рабочую директорию	8
2.4	Вставка кода из файла листинга 9.1	8
2.5	Сборка программы из файла lab9-1.asm и её запуск	9
2.6	Изменение файла lab9-1.asm	9
2.7	Повторная сборка программы из файла lab9-1.asm и её запуск	9
2.8	Создание второго файла: lab9-2.asm	10
2.9	Запись кода из листинга 9.2 в файл lab9-2.asm	10
2.10	Сборка программы из файла lab9-2.asm	10
2.11	Загрузка программы lab9-2.asm в gdb	11
2.12	Запуск программы в отладчике	11
2.13	Создание брейкпоинта	11
2.14	Дизассемблирование программы	12
2.15	Переключение на синтаксис intel	12
2.16	Повторное дизассемблирование	12
2.17	Внешний вид интерфейса	13
2.18	Вывод информации о брейкпоинтах	13
2.19	Создание брейкпоинта по адресу	14
2.20	Повторный вывод информации о брейкпоинтах	14
2.21	Выполнение следующей команды в коде программы	14
2.22	Выполнение следующей команды в коде программы (2)	15
2.23	Вывод значений регистров	15
2.24	Значения регистров	15
2.25	Вывод значения переменной по имени	16
2.26	Вывод значения переменной по адресу	16
2.27	Изменение первого символа переменной по имени и вывод переменной	16
2.28	Изменение второго символа переменной по адресу и вывод переменной	16
2.29	Изменение нескольких символов второй переменной по адресу и вывод переменной	17
2.30	Вывод значения регистра в строковом, двоичном и шестнадцатичном виде	17
2.31	Изменение значения регистра	17
2.32	Завершение работы программы	18
2.33	Копирование файла из прошлой работы	18

2.34 Сборка программы и загрузка в gdb	18
2.35 Создание брейкпоинта и запуск программы	18
2.36 Вывод значения регистра esp	19
2.37 Вывод всех значений в стеке	19
3.1 Копирование первого файла самостоятельной работы из прошлой работы	20
3.2 Редактирование кода	21
3.3 Сборка и проверка работы программы	21
3.4 Создание файла второго задания самостоятельной работы	22
3.5 Вставка кода из листинга 9.3	22
3.6 Сборка программы	22
3.7 Переключение на синтаксис intel	23
3.8 Включение графического отображения кода и выполнения команд	23
3.9 Включение графического отображения значений регистров	23
3.10 Установка брейкпоинта	23
3.11 Значение всех регистров на 1 шаге	24
3.12 Значение всех регистров на 2 шаге	25
3.13 Значение всех регистров на 3 шаге	26
3.14 Значение всех регистров на 4 шаге	26
3.15 Значение всех регистров на 5 шаге	27
3.16 Значение всех регистров на 6 шаге	27
3.17 Редактирование кода	28
3.18 Сборка кода и проверка выполнения	28

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 выполнения работы необходимо создать рабочую папку и файл lab9-1.asm :

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

Рис. 2.1: Создание рабочей директории и файла lab9-1.asm

Далее, запустим Midnight commander :

```
rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$ mc
```

Рис. 2.2: Запуск Midnight commander

Скопируем файл in_out.asm из директории прошлой работы :

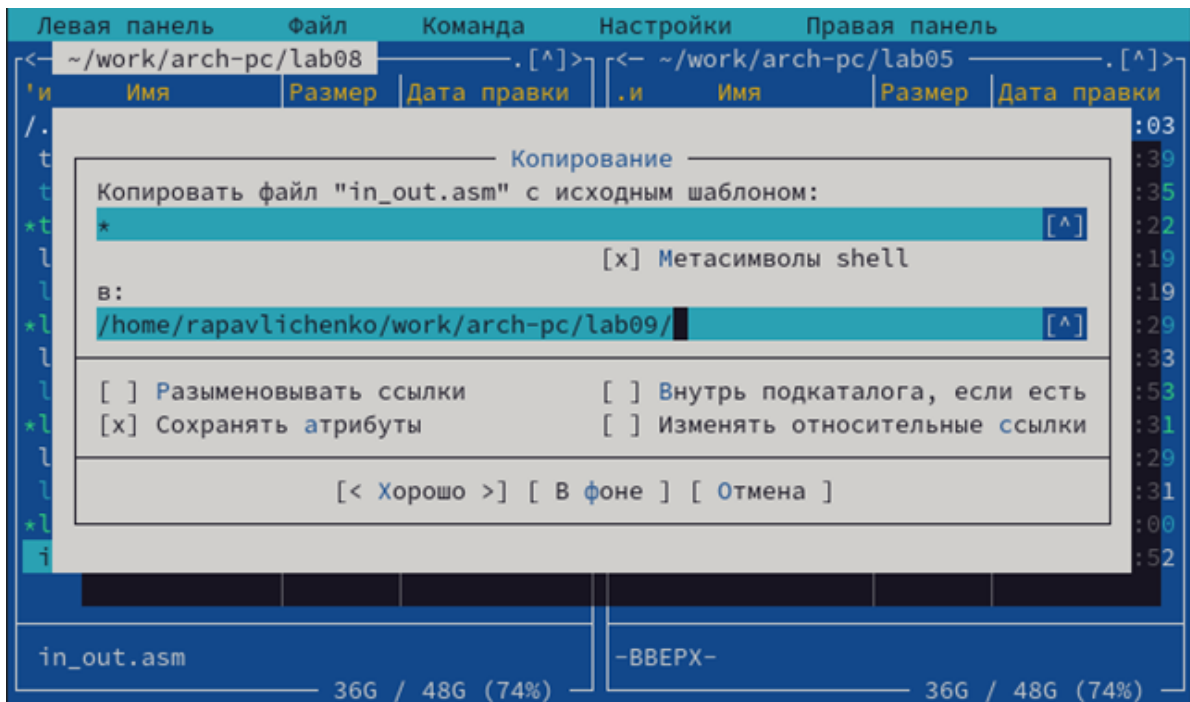


Рис. 2.3: Копирование файла in_out.asm в рабочую директорию

Вставим в файл lab9-1.asm код из листинга 9.1 :

```
GNU nano 7.2 /home/rapavlchenko/work/arch-pc/lab09/lab09-1.asm  Изменён
%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
```

Рис. 2.4: Вставка кода из файла листинга 9.1

Соберём программу и посмотрим на вывод :

```
rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 15
2x+7=37
rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$
```

Рис. 2.5: Сборка программы из файла lab9-1.asm и её запуск

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 $f(g(x))$:

```
GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab09/lab09-1.asm
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(g(x))=',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
```

Рис. 2.6: Изменение файла lab9-1.asm

Соберём программу и проверим её работу :

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

Рис. 2.7: Повторная сборка программы из файла lab9-1.asm и её запуск

Создадим новый файл lab9-2.asm:

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

Рис. 2.8: Создание второго файла: lab9-2.asm

Вставим в него код из листинга 9.2 :

```
GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab09/lab09-2.asm
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
```

Рис. 2.9: Запись кода из листинга 9.2 в файл lab9-2.asm

Соберём программу следующим образом (с использованием аргумента -g) :

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

Рис. 2.10: Сборка программы из файла lab9-2.asm

Теперь загрузим её в gdb :

```

rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 14.2-1.fc40
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 lab09-2...
(gdb)

```

Рис. 2.11: Загрузка программы lab9-2.asm в gdb

Запустим её в отладчике с помощью команды run :

```

(gdb) run
Starting program: /home/rapavlichenko/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 15597) exited normally]
(gdb)

```

Рис. 2.12: Запуск программы в отладчике

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/rapavlichenko/work/arch-pc/lab09/lab09-2

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

```

Рис. 2.13: Создание брейкпоинта

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. 2.14: Дизассемблирование программы

Переключим синтаксис вывода на intel :

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

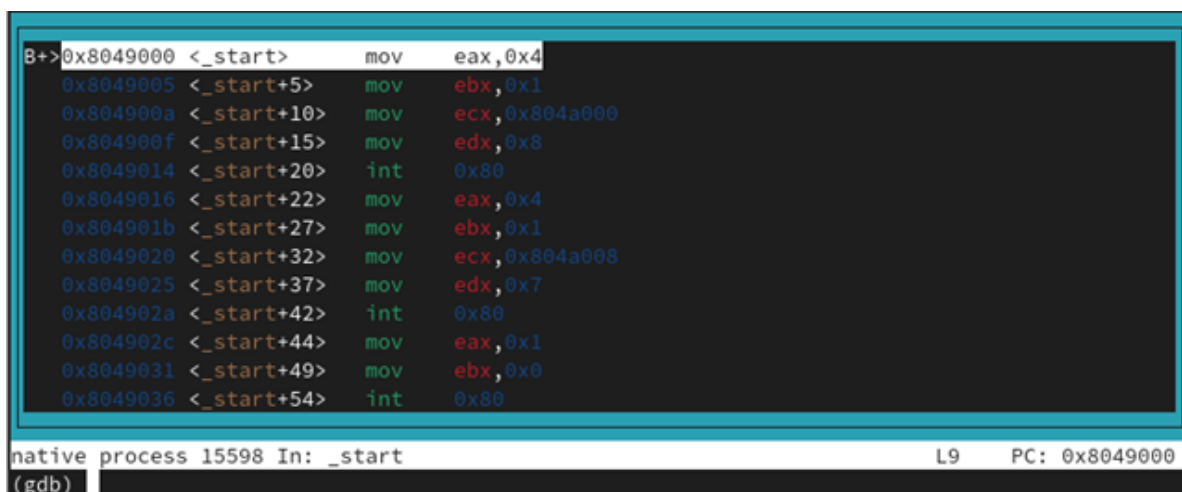
Рис. 2.15: Переключение на синтаксис intel

Повторно дизассемблируем программу :

```
(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. 2.16: Повторное дизассемблирование

Включим графическое отображения кода :

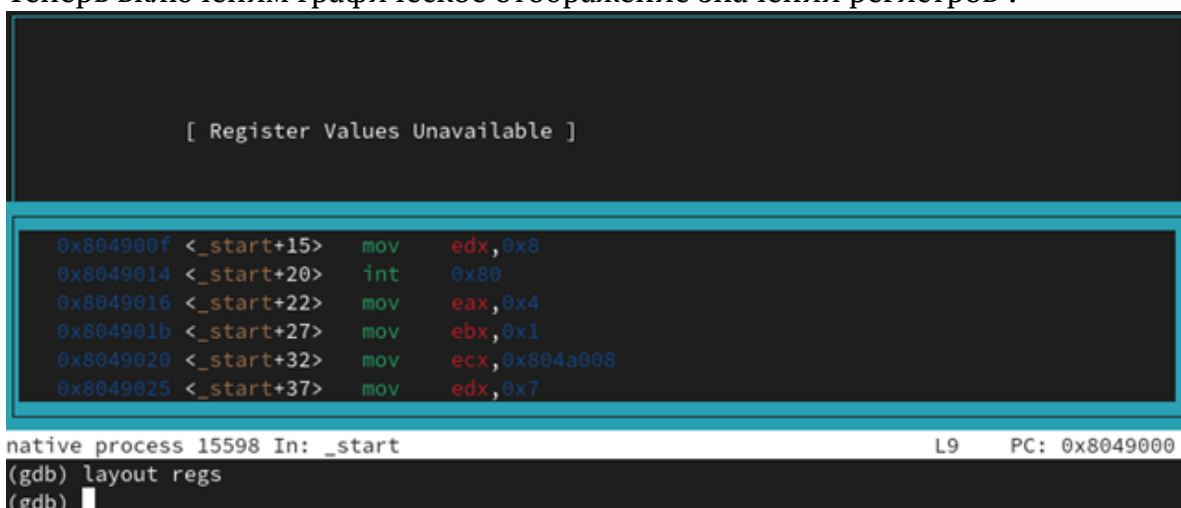


```
B+>0x8049000 <_start>    mov     eax,0x4
0x8049005 <_start+5>    mov     ebx,0x1
0x804900a <_start+10>   mov     ecx,0x804a000
0x804900f <_start+15>   mov     edx,0x8
0x8049014 <_start+20>   int     0x80
0x8049016 <_start+22>   mov     eax,0x4
0x804901b <_start+27>   mov     ebx,0x1
0x8049020 <_start+32>   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 15598 In: _start          L9      PC: 0x8049000
(gdb)
```

Рис. 2.17: Внешний вид интерфейса

Теперь включим графическое отображение значений регистров :

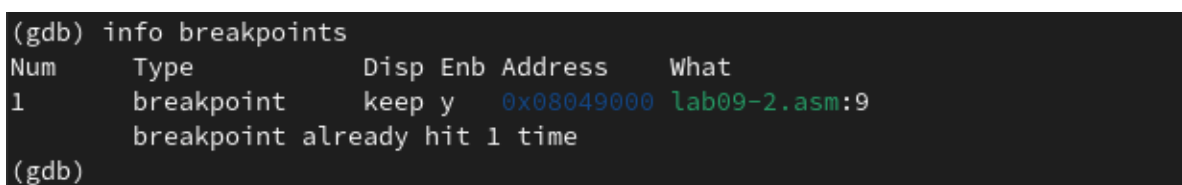


```
[ Register Values Unavailable ]

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 15598 In: _start          L9      PC: 0x8049000
(gdb) layout regs
(gdb)
```

Выведем информацию о всех брейкпоинтах :



```
(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. 2.18: Вывод информации о брейкпоинтах

Попробуем теперь создать брейкпоинт по адресу:

```

native process 15598 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb)

```

Рис. 2.19: Создание брейкпоинта по адресу

Повторно выведем информацию о брейкпоинтах :

```

(gdb) info breakpoints
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: Повторный вывод информации о брейкпоинтах

Теперь 5 раз выполним команду si для построчного выполнения кода :

```

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 15653 In: _start L9 PC: 0x8049000
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/rapavlichenko/work/arch-pc/lab09/lab09-2

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

```

Рис. 2.21: Выполнение следующей команды в коде программы

```

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

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

native process 15653 In: _start L14 PC: 0x8049016

Breakpoint 1, _start () at lab09-2.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.22: Выполнение следующей команды в коде программы (2)

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

```
(gdb) info registers
```

Рис. 2.23: Вывод значений регистров

Вот, что нам выводится :

```

native process 15653 In: _start L14 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd080 0xffffd080
ebp      0x0      0x0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 2.24: Значения регистров

Попробуем вывести значени переменной по имени :

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

Рис. 2.25: Вывод значения переменной по имени

Теперь попробуем вывести значени переменной по адресу :

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

Рис. 2.26: Вывод значения переменной по адресу

Теперь изменим первый символ переменной :

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

Рис. 2.27: Изменение первого символа переменной по имени и вывод переменной

А теперь изменим второй символ переменной, уже обратясь по адресу :

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

Рис. 2.28: Изменение второго символа переменной по адресу и вывод переменной

Теперь изменим несколько символов второй переменной :


```
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804aa00b=' '
Cannot access memory at address 0x804aa00b
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb &msg2
0x804a008 <msg2>: "Lor d!\n\034"
(gdb)
```

Рис. 2.29: Изменение нескольких символов второй переменной по адресу и вывод переменной

Теперь попробуем вывести значение регистра в строковом, двоичном и шестнадцатичном виде :

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

Рис. 2.30: Вывод значения регистра в строковом, двоичном и шестнадцатичном виде

Попробуем теперь изменить значение регистра :

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb)
```

Рис. 2.31: Изменение значения регистра

Как видим, в регистр записались разные значения. Это связано с тем, что в одном случае мы записываем в него число, а в другом случае - строку. Завершим работу программы с помощью `continue` (чтобы продолжить выполнение) и выйдем из отладчика :

```
(gdb) continue
Continuing.
Lor d!

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

Рис. 2.32: Завершение работы программы

Скопируем файл из прошлой работы :

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

Рис. 2.33: Копирование файла из прошлой работы

Соберём его и вгрузим в gdb :

```
rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
```

Рис. 2.34: Сборка программы и загрузка в gdb

Создадим брейкпоинт и запустим программу :

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/rapavlichenko/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. 2.35: Создание брейкпоинта и запуск программы

Теперь выведем значение регистра есп, где хранятся данные о стеке :

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

Рис. 2.36: Вывод значения регистра esp

Теперь выведем значение всех элементов стека :

```
(gdb) x/s *(void**)( $esp + 4)
0xffffd1f9:      "/home/rapavlichenko/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xffffd228:      "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xffffd23a:      "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xffffd24b:      "2"
(gdb) x/s *(void**)( $esp + 20)
0xffffd24d:      "аргумент 3"
(gdb) x/s *(void**)( $esp + 24)
0x0:      <error: Cannot access memory at address 0x0>
(gdb) █
```

Рис. 2.37: Вывод всех значений в стеке

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, ведь под каждый элемент выделяется 4 байта

3 Выполнение задания для самостоятельной работы

Скопируем файл первого задания прошлой самостоятельной работы :

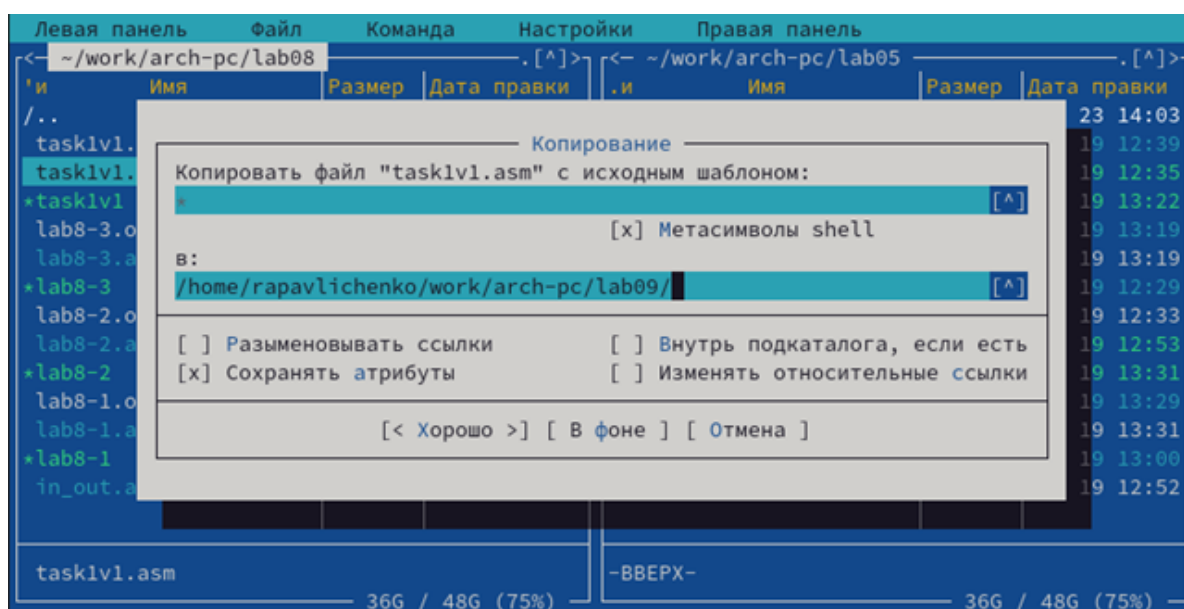


Рис. 3.1: Копирование первого файла самостоятельной работы из прошлой работы

Нам нужно переписать его так, чтобы он использовал для авчисления выражения подпрограмму :

```
GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab09/task1v1.asm Изменён
#include 'in_out.asm'
SECTION .data
msg db "Результат: ", 0
msg2 db "Функция: f(x) = 2x + 15", 0
SECTION .text
global _start
_start:
pop ecx          ; Извлекаем количество аргументов
pop edx          ; Извлекаем имя программы
sub ecx, 1       ; Уменьшаем ecx (только аргументы, без имени программы)
mov esi, 0       ; Обнуляем промежуточную сумму
next:
cmp ecx, 0h      ; Проверяем, остались ли аргументы
jz _end          ; Если нет, переходим к завершению программы
pop eax          ; Извлекаем следующий аргумент из стека
call atoi        ; Преобразуем строку в число
call _calcul     ; Вызываем подпрограмму вычисления f(x)
add esi, eax      ; Добавляем результат к промежуточной сумме
loop next        ; Переход к обработке следующего аргумента
_end:
mov eax, msg2     ; Выводим описание функции
call sprintf      ; Выводим сообщение "Результат: "
mov eax, esi      ; Передаем результат в регистр eax
call iprintLF     ; Выводим результат
call quit        ; Завершаем программу
_calcul:
mov ebx, 2        ; Подготовка множителя для функции f(x)
mul ebx           ; eax = eax * 2
add eax, 15       ; eax = eax + 15
ret              ; Возврат из подпрограммы
```

Рис. 3.2: Редактирование кода

Соберём его и проверим корректность выполнения :

```
rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$ nasm -f elf task1v1.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$ ld -m elf_i386 -o task1v1 task1v1.o
rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$ ./task1v1 1 2 3 4
Функция: f(x) = 2x + 15
Результат: 80
rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$ 5 6 7 8
bash: 5: команда не найдена...
rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$ ./task1v1 5 6 7 8
Функция: f(x) = 2x + 15
Результат: 112
rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$
```

Рис. 3.3: Сборка и проверка работы программы

Создадим файл второго задания самостоятельной работы :

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

Рис. 3.4: Создание файла второго задания самостоятельной работы

Вставим в него код из листинга 9.3 :

```
GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab09/task2.asm
%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: Вставка кода из листинга 9.3

Соберём его и запустим:

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

Рис. 3.6: Сборка программы

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

```

rapavlichenko@rapavlichenko:~/work/arch-pc/lab09$ gdb task2
GNU gdb (Fedora Linux) 14.2-1.fc40
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 task2...
(gdb)

```

Переключим его на синтаксис intel :

```

(gdb) set disassembly-flavor intel
(gdb)

```

Рис. 3.7: Переключение на синтаксис intel

Включим графическое отображение кода :

```

(gdb) layout asm

```

Рис. 3.8: Включение графического отображения кода и выполнения команд

Включим графическое отображение значений регистров :

```

(gdb) layout regs

```

Рис. 3.9: Включение графического отображения значений регистров

Установим брейкпоинт на `_start` :

```

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

```

Рис. 3.10: Установка брейкпоинта

И начнём построчно выполнять код :

```
native process 19474 In: _start L9 PC: 0x80490ed
Breakpoint 1 at 0x80490e8: file task2.asm, line 8.
(gdb) run
Starting program: /home/rapavlichenko/work/arch-pc/lab09/task2

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 task2.asm:8
(gdb) si
(gdb) █
```

Рис. 3.11: Значение всех регистров на 1 шаге


```
Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd080 0xffffd080
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f2 0x80490f2 <_start+10>
eflags   0x10202 [ IF RF ]
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 <iprintf>
0x8049111 <_start+41>      call    0x80490db <quit>

native process 19474 In: _start L10 PC: 0x80490f2
(gdb) run
Starting program: /home/rapavlichenko/work/arch-pc/lab09/task2

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 task2.asm:8
(gdb) si
(gdb) si
(gdb) |
```

Рис. 3.12: Значение всех регистров на 2 шаге

```
Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x5      5
esp      0xffffd080 0xffffd080
ebp      0x0      0x0
esi      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

native process 19474 In: _start L11 PC: 0x80490f4
(gdb) si
(gdb) █
```

Рис. 3.13: Значение всех регистров на 3 шаге

```
Register group: general
eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd080 0xffffd080
ebp      0x0      0x0
esi      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

native process 19474 In: _start L12 PC: 0x80490f9
(gdb) si
(gdb) si
(gdb) si
```

Рис. 3.14: Значение всех регистров на 4 шаге

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd080 0xffffd080
ebp      0x0      0x0
esi      0x0      0

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>

native process 19474 In: _start L13 PC: 0x80490fb
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 3.15: Значение всех регистров на 5 шаге

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd080 0xffffd080
ebp      0x0      0x0
esi      0x0      0

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>

native process 19474 In: _start L14 PC: 0x80490fe
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 3.16: Значение всех регистров на 6 шаге

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 /home/rapavlichenko/work/arch-pc/lab09/task2.asm
%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: Редактирование кода

И проверим корректность его выполнения :

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

Рис. 3.18: Сборка кода и проверка выполнения

Как видим, теперь код работает корректно

4 Выводы

В результате выполнения лабораторной работы были получены представления о работе подпрограмм, а также было реализовано несколько программ, использующих подпрограммы. Также, были получены навыки работы с базовым функционалом gdb, и с помощью gdb была отловлена ошибка в коде программы