

Лабораторная работа №9

Понятие подпрограммы. Отладчик GDB

Андрюшин Никита Сергеевич

Содержание

| | | |
|---|---|----|
| 1 | Цель работы | 6 |
| 2 | Выполнение лабораторной работы | 7 |
| 3 | Выполнение задания для самостоятельной работы | 24 |
| 4 | Выводы | 37 |

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 | 9 |
| 2.5 | Сборка программы из файла lab9-1.asm и её запуск | 9 |
| 2.6 | Изменение файла lab9-1.asm | 10 |
| 2.7 | Повторная сборка программы из файла lab9-1.asm и её запуск . . | 10 |
| 2.8 | Создание второго файла: lab9-2.asm | 11 |
| 2.9 | Запись кода из листинга 9.2 в файл lab9-2.asm | 11 |
| 2.10 | Сборка программы из файла lab9-2.asm | 11 |
| 2.11 | Загрузка программы lab9-2.asm в gdb | 12 |
| 2.12 | Запуск программы в отладчике | 12 |
| 2.13 | Создание брейкпоинта | 12 |
| 2.14 | Дизассемблирование программы | 13 |
| 2.15 | Переключение на синтаксис intel | 13 |
| 2.16 | Повторное дизассемблирование | 13 |
| 2.17 | Включение графического отображения кода и выполнения команд | 14 |
| 2.18 | Внешний вид интерфейса | 14 |
| 2.19 | включение графического отображения значений регистров и отображение интерфейса | 15 |
| 2.20 | Вывод информации о брейкпоинтах | 15 |
| 2.21 | Создание брейкпоинта по адресу | 16 |
| 2.22 | Повторный вывод информации о брейкпоинтах | 16 |
| 2.23 | Выполнение следующей команды в коде программы (1) | 17 |
| 2.24 | Выполнение следующей команды в коде программы (2) | 17 |
| 2.25 | Выполнение следующей команды в коде программы (3) | 18 |
| 2.26 | Выполнение следующей команды в коде программы (4) | 18 |
| 2.27 | Выполнение следующей команды в коде программы (5) | 19 |
| 2.28 | Вывод значений регистров | 19 |
| 2.29 | Значения регистров | 19 |
| 2.30 | Вывод значения переменной по имени | 20 |
| 2.31 | Вывод значения переменной по адресу | 20 |
| 2.32 | Изменение первого символа переменной по имени и вывод переменной | 20 |
| 2.33 | Изменение второго символа переменной по адресу и вывод переменной | 20 |

| | | |
|------|--|----|
| 2.34 | Изменение нескольких символов второй переменной по адресу и вывод переменной | 21 |
| 2.35 | Вывод значения регистра в строковом, двоичном и шестнадцатичном виде | 21 |
| 2.36 | Изменение значения регистра | 21 |
| 2.37 | Завершение работы программы | 22 |
| 2.38 | Завершение работы программы (продолжение) | 22 |
| 2.39 | Копирование файла из прошлой работы | 22 |
| 2.40 | Сборка программы и загрузка в gdb | 22 |
| 2.41 | Создание брейкпоинта и запуск программы | 23 |
| 2.42 | Вывод значения регистра esp | 23 |
| 2.43 | Вывод всех значений в стеке | 23 |
| 3.1 | Копирование первого файла самостоятельной работы из прошлой работы | 24 |
| 3.2 | Редактирование кода | 25 |
| 3.3 | Редактирование кода (продолжение) | 25 |
| 3.4 | Сборка и проверка работы программы | 26 |
| 3.5 | Создание файла второго задания самостоятельной работы | 26 |
| 3.6 | Вставка кода из листинга 9.3 | 26 |
| 3.7 | Сборка программы | 27 |
| 3.8 | Запуск программы | 27 |
| 3.9 | Загрузка программы в gdb | 27 |
| 3.10 | Переключение на синтаксис intel | 27 |
| 3.11 | Включение графического отображения кода и выполнения команд | 28 |
| 3.12 | Включение графического отображения значений регистров | 28 |
| 3.13 | Установка брейкпоинта | 28 |
| 3.14 | Значение всех регистров на 0 шаге | 29 |
| 3.15 | Значение всех регистров на 1 шаге | 30 |
| 3.16 | Значение всех регистров на 2 шаге | 31 |
| 3.17 | Значение всех регистров на 3 шаге | 32 |
| 3.18 | Значение всех регистров на 4 шаге | 33 |
| 3.19 | Значение всех регистров на 5 шаге | 34 |
| 3.20 | Значение всех регистров на 6 шаге | 35 |
| 3.21 | Редактирование кода | 36 |
| 3.22 | Сборка кода и проверка выполнения | 36 |

Список таблиц

1 Цель работы

Ознакомиться с понятием подпрограмм в Ассемблере и научиться использовать подпрограммы на практике. Ознакомиться с отладчиком gdb и научиться использовать его

2 Выполнение лабораторной работы

Для начала выполнения работы необходимо создать рабочую папку и файл lab9-1.asm (Рис. 2.1):

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

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

Далее, запустим Midnight commander (Рис. 2.2):

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

Рис. 2.2: Запуск Midnight commander

Скопируем файл in_out.asm из директории прошлой работы (Рис. 2.3):

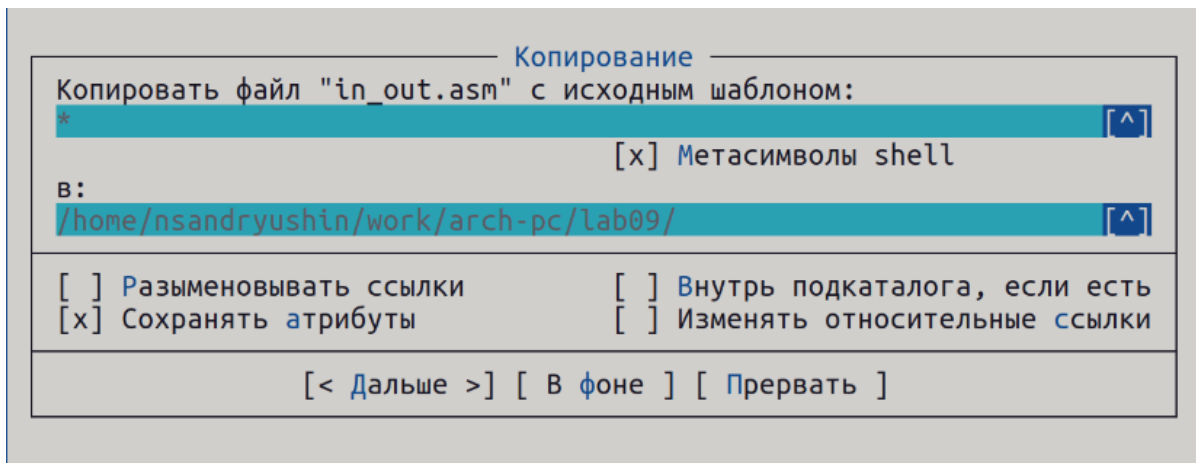


Рис. 2.3: Копирование файла in_out.asm в рабочую директорию

Вставим в файл lab9-1.asm код из листинга 9.1 (Рис. 2.4):


```

GNU nano 6.2 /home/nsandryushin/work/arch
%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 (Рис. 2.5):

```

nsandryushin@nsandryushin:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
nsandryushin@nsandryushin:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
nsandryushin@nsandryushin:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 10
2x+7=27
nsandryushin@nsandryushin:~/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 $f(g(x))$ (Рис. 2.6):

```
GNU nano 6.2 /home/nsandryushin/work/arch-pc/lab09/lab09-1.asm
mov eax,[res]
call iprintLF
call quit
;-----
; 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. 2.6: Изменение файла lab9-1.asm

Соберём программку и проверим её работу (Рис. 2.7):

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

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

Создадим новый файл (Рис. 2.8):

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

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

Вставим в него код из листинга 9.2 (Рис. 2.9):

```
GNU nano 6.2 /home/nsandryushin/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) (Рис. 2.10):

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

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

Теперь загрузим её в gdb (Рис. 2.11):

```

nsandryushin@nsandryushin:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-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 (Рис. 2.12):

```

(gdb) run
Starting program: /home/nsandryushin/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 3590) exited normally]
(gdb)

```

Рис. 2.12: Запуск программы в отладчике

Создадим брейкпоинт на метке _start с помощью команды break (Рис. 2.13):

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/nsandryushin/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 (Рис. 2.14):

```
(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 (Рис. 2.15):

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

Рис. 2.15: Переключение на синтаксис intel

Повторно дизассемблируем программу (Рис. 2.16):

```
(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)
```

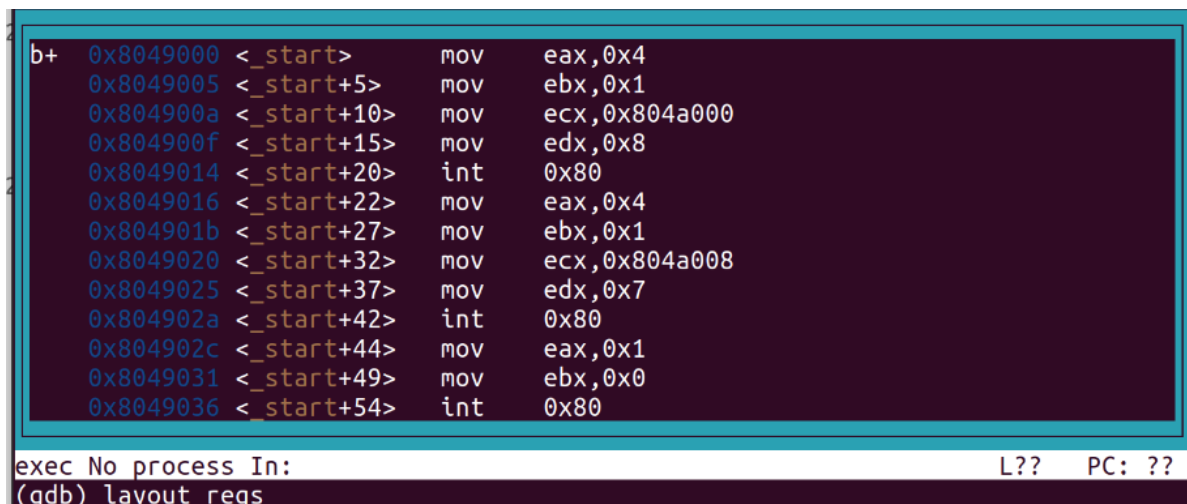
Рис. 2.16: Повторное дизассемблирование

Включим графическое отображения кода (Рис. 2.17):

```
(gdb) layout asm
```

Рис. 2.17: Включение графического отображения кода и выполнения команд

Вот как теперь это выглядит (Рис. 2.18):



```
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
```

exec No process In: L?? PC: ??

(gdb) layout regs

Рис. 2.18: Внешний вид интерфейса

Теперь включим графическое отображение значений регистров (Рис. 2.19):

```
[ 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

exec No process in: L?? PC: ??
(gdb) layout regs
(gdb)
```

Рис. 2.19: включение графического отображения значений регистров и отображение интерфейса

Выведем информацию о всех брейкпоинтах (Рис. 2.20):

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

Рис. 2.20: Вывод информации о брейкпоинтах

Попробуем теперь создать брейкпоинт по адресу (Рис. 2.21):

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

exec No process in:                               L??  PC: ??
(gdb) layout regs
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab09-2.asm:9
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb)
```

Рис. 2.21: Создание брейкпоинта по адресу

Повторно выведем информацию о брейкпоинтах (Рис. 2.22):

```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab09-2.asm:9
2        breakpoint     keep y   0x08049031 lab09-2.asm:20
(gdb)
```

Рис. 2.22: Повторный вывод информации о брейкпоинтах

Теперь 5 раз выполним команду `si` для построчного выполнения кода (Рис. 2.23 - 2.27):


```
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1b0 0xffffd1b0

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

native process 3946 In: _start L10 PC: 0x8049005
(gdb) si
The program is not being run.
(gdb) run
Starting program: /home/nsandryushin/work/arch-pc/lab09/lab09-2

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

Рис. 2.23: Выполнение следующей команды в коде программы (1)

```
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0

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

native process 3946 In: _start L11 PC: 0x804900a
The program is not being run.
(gdb) run
Starting program: /home/nsandryushin/work/arch-pc/lab09/lab09-2

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

Рис. 2.24: Выполнение следующей команды в коде программы (2)

```
eax      0x4      4
ecx      0x804a000 134520832
edx      0x0      0
ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0

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

native process 3946 In: _start L12 PC: 0x804900f
(gdb) run
Starting program: /home/nsandryushin/work/arch-pc/lab09/lab09-2

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

Рис. 2.25: Выполнение следующей команды в коде программы (3)

```
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0

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

native process 3946 In: _start L13 PC: 0x8049014
Starting program: /home/nsandryushin/work/arch-pc/lab09/lab09-2

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

Рис. 2.26: Выполнение следующей команды в коде программы (4)

```
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0

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 3946 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
```

Рис. 2.27: Выполнение следующей команды в коде программы (5)

Как видим, поменялись значения регистров `eax`, `ecx`, `edx` и `ebx`. Теперь выведем информацию о значениях регистров (Рис. 2.28):

```
(gdb) info registers
```

Рис. 2.28: Вывод значений регистров

Вот, что нам выводится (Рис. 2.29):

```
native process 3946 In: _start L14 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 2.29: Значения регистров

Попробуем вывести значения переменной по имени (Рис. 2.30):

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

Рис. 2.30: Вывод значения переменной по имени

Теперь попробуем вывести значени переменной по адресу (Рис. 2.31):

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

Рис. 2.31: Вывод значения переменной по адресу

Теперь изменим первый символ переменной (Рис. 2.32):

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

Рис. 2.32: Изменение первого символа переменной по имени и вывод переменной

А теперь изменим второй символ переменной, уже обратясь по адресу (Рис. 2.33):

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

Рис. 2.33: Изменение второго символа переменной по адресу и вывод переменной

Теперь изменим несколько символов второй переменной (Рис. 2.34):

```
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb &msg2
0x804a008 <msg2>: "Lor d!\n\034"
(gdb) █
```

Рис. 2.34: Изменение нескольких символов второй переменной по адресу и вывод переменной

Теперь попробуем вывести значение регистра в строковом, двоичном и шестнадцатичном виде (Рис. 2.35):

```
(gdb) print /s $edx
$3 = 8
(gdb) print /t $edx
$4 = 1000
(gdb) print /x $edx
$5 = 0x8
(gdb) █
```

Рис. 2.35: Вывод значения регистра в строковом, двоичном и шестнадцатичном виде

Попробуем теперь изменить значение регистра (Рис. 2.36):

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$6 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$7 = 2
(gdb) █
```

Рис. 2.36: 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 (Рис. 2.37 - 2.38):

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

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

Рис. 2.37: Завершение работы программы

```
(gdb) continue
Continuing.
[Inferior 1 (process 4295) exited normally]
(gdb) q
```

Рис. 2.38: Завершение работы программы (продолжение)

Скопируем файл из прошлой работы (Рис. 2.39):

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

Рис. 2.39: Копирование файла из прошлой работы

Соберём его и вгрузим в gdb (Рис. 2.40):

```
nsandryushin@nsandryushin:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab0
9-3.asm
nsandryushin@nsandryushin:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.
o
nsandryushin@nsandryushin:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргум
ент 2 'аргумент 3'
```

Рис. 2.40: Сборка программы и загрузка в gdb

Создадим брейкпоинт и запустим программу (Рис. 2.41):

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

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

Рис. 2.41: Создание брейкпоинта и запуск программы

Теперь выведем значение регистра esp, где хранятся данные о стеке (Рис. 2.42):

```
(gdb) x/x $esp
0xffffd170: 0x00000005
```

Рис. 2.42: Вывод значения регистра esp

Теперь выведем значение всех элементов стека (Рис. 2.43):

```
(gdb) x/s *(void**)(esp + 4)
0xffffd322: "/home/nsandryushin/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd350: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd362: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd373: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd375: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.43: 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):

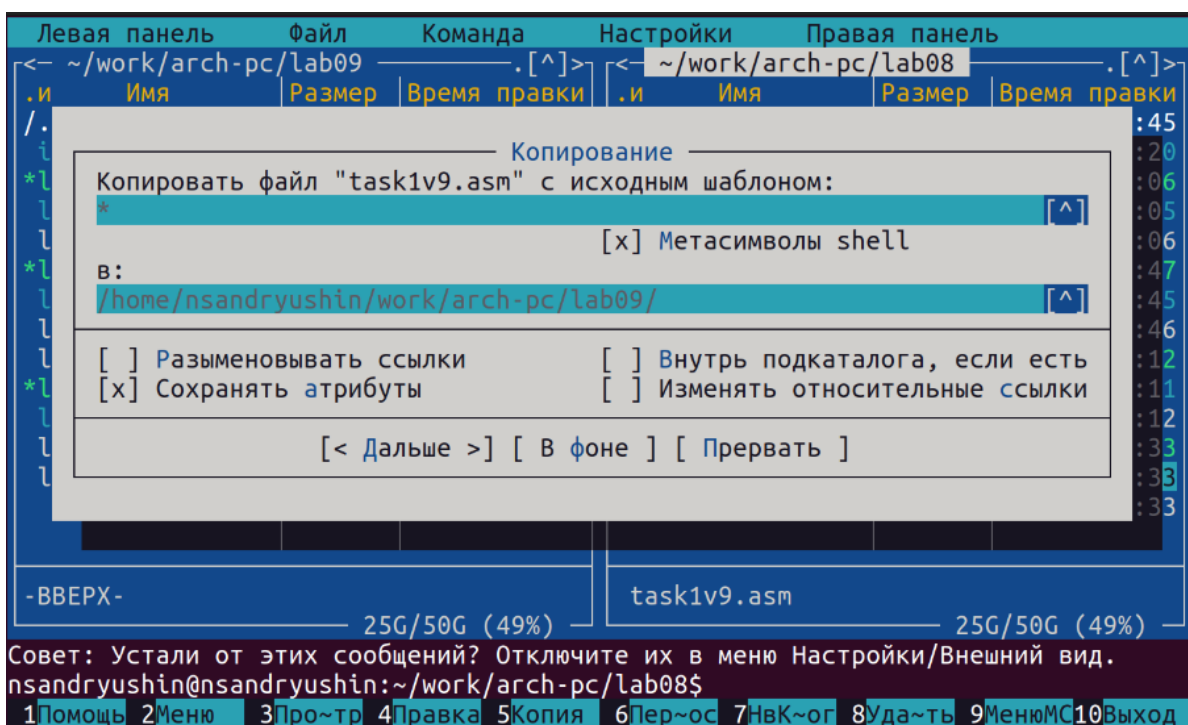


Рис. 3.1: Копирование первого файла самостоятельной работы из прошлой работы

Нам нужно переписать его так, чтобы он использовал для авчисления выражения подпрограмму (Рис. 3.2 - 3.3):


```

GNU nano 6.2 /home/nsandryushin/work/arch-pc/lab09/task1v9.asm
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg2 db "Функция: f(x)=10x-4",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
mov esi, 0 ; Используем `esi` для хранения
; промежуточных сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека

```

Рис. 3.2: Редактирование кода

```

GNU nano 6.2 /home/nsandryushin/work/arch-pc/lab09/task1v9.asm
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
call _calcul
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg2
call sprintf
mov eax, msg ; вывод сообщения "Результат: "
call sprintf
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
_calcul:
mov ebx, 10
mul ebx
sub eax, 4
ret

```

Рис. 3.3: Редактирование кода (продолжение)

Соберём его и проверим корректность выполнения (Рис. 3.4):

```

nsandryushin@nsandryushin:~/work/arch-pc/lab09$ nasm -f elf task1v9.asm
nsandryushin@nsandryushin:~/work/arch-pc/lab09$ ld -m elf_i386 -o task1v9 task1v9.o
nsandryushin@nsandryushin:~/work/arch-pc/lab09$ ./task1v9 1 2 3 4
Функция:  $f(x)=10x-4$ 
Результат: 84
nsandryushin@nsandryushin:~/work/arch-pc/lab09$ ./task1v9 1
Функция:  $f(x)=10x-4$ 
Результат: 6
nsandryushin@nsandryushin:~/work/arch-pc/lab09$ ./task1v9 1 2
Функция:  $f(x)=10x-4$ 
Результат: 22

```

Рис. 3.4: Сборка и проверка работы программы

Создадим файл второго задания самостоятельной работы (Рис. 3.5):

```

nsandryushin@nsandryushin:~/work/arch-pc/lab09$ touch task2.asm
nsandryushin@nsandryushin:~/work/arch-pc/lab09$ █

```

Рис. 3.5: Создание файла второго задания самостоятельной работы

Вставим в него код из листинга 9.3 (Рис. 3.6):

```

GNU nano 6.2 /home/nsandryushin/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. 3.6: Вставка кода из листинга 9.3

Соберём его (Рис. 3.7):

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

Рис. 3.7: Сборка программы

И запустим (Рис. 3.8):

```
nsandryushin@nsandryushin:~/work/arch-pc/lab09$ ./task2
Результат: 10
nsandryushin@nsandryushin:~/work/arch-pc/lab09$
```

Рис. 3.8: Запуск программы

Как видим, код считает значение выражения неправильно. Загрузим его в gdb (Рис. 3.9):

```
nsandryushin@nsandryushin:~/work/arch-pc/lab09$ gdb task2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-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. 3.9: Вгрузка программы в gdb

Переключим его на синтаксис intel (Рис. 3.10):

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

Рис. 3.10: Переключение на синтаксис intel

Включим графическое отображение кода (Рис. 3.11):

```
(gdb) layout asm
```

Рис. 3.11: Включение графического отображения кода и выполнения команд

Включим графическое отображение значений регистров (Рис. 3.12):

```
(gdb) layout regs
```

Рис. 3.12: Включение графического отображения значений регистров

Установим брейкпоинт на `_start` (Рис. 3.13):

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

Рис. 3.13: Установка брейкпоинта

И начнём построчно выполнять код (Рис. 3.14 - 3.20):

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd110 0xffffd110

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 3644 In: _start L8 PC: 0x80490e8
(gdb) layout regs
(gdb) break _start
Breakpoint 1 at 0x80490e8: file task2.asm, line 8.
(gdb) run
Starting program: /home/nsandryushin/work/arch-pc/lab09/task2

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

Рис. 3.14: Значение всех регистров на 0 шаге

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd110 0xffffd110

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 3644 In: _start L9 PC: 0x80490ed
(gdb) break _start
Breakpoint 1 at 0x80490e8: file task2.asm, line 8.
(gdb) run
Starting program: /home/nsandryushin/work/arch-pc/lab09/task2

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

Рис. 3.15: Значение всех регистров на 1 шаге

```
Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd110 0xffffd110

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 3644 In: _start L10 PC: 0x80490f2
Breakpoint 1 at 0x80490e8: file task2.asm, line 8.
(gdb) run
Starting program: /home/nsandryushin/work/arch-pc/lab09/task2

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

Рис. 3.16: Значение всех регистров на 2 шаге

```
Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x5      5
esp      0xffffd110 0xffffd110

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 3644 In: _start L11 PC: 0x80490f4
(gdb) run
Starting program: /home/nsandryushin/work/arch-pc/lab09/task2

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

Рис. 3.17: Значение всех регистров на 3 шаге


```
Register group: general
eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd110 0xffffd110

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 3644 In: _start L12 PC: 0x80490f9
Starting program: /home/nsandryushin/work/arch-pc/lab09/task2

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

Рис. 3.18: Значение всех регистров на 4 шаге

```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd110 0xffffd110

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 3644 In: _start L13 PC: 0x80490fb

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

Рис. 3.19: Значение всех регистров на 5 шаге

```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd110 0xffffd110

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 3644 In: _start L14 PC: 0x80490fe
Breakpoint 1, _start () at task2.asm:8
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рис. 3.20: 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 (Рис. 3.21):

```

GNU nano 6.2 /home/nsandryushin/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. 3.21: Редактирование кода

И проверим корректность его выполнения (Рис. 3.22):

```

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

```

Рис. 3.22: 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