

Лабораторная работа №9

Понятие подпрограммы. Отладчик GDB.

Чекмарев Александр Дмитриевич | группа: НПИбд 02-23

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
2.1	Реализация подпрограмм в NASM	7
2.2	Отладка программ с помощью GDB	11
2.3	Добавление точек останова	16
2.4	Работа с данными программы в GDB	18
2.5	Обработка аргументов командной строки в GDB	25
3	Самостоятельная работа	28
4	Выводы	34

Список иллюстраций

2.1	Рис 2.1.1: Создание каталога и файла .asm	7
2.2	Рис 2.1.2: Демонстрация текста программы в файле	8
2.3	Рис 2.1.3: Создание файла и его проверка	9
2.4	Рис 2.1.4: Демонстрация измененного текста программы	10
2.5	Рис 2.1.5: Создание файла и его проверка	11
2.6	Рис 2.2.1: Создание файла .asm	11
2.7	Рис 2.2.2: Демонстрация текста программы	12
2.8	Рис 2.2.3: Получение испол. файла с ключом -g	12
2.9	Рис 2.2.4: Загрузка испол. файла в отладчик	13
2.10	Рис 2.2.5: Проверка работы файла	13
2.11	Рис 2.2.6: Установка брейкпоинта на метку и запуск программы .	13
2.12	Рис 2.2.7: Использование команды disassemble	14
2.13	Рис 2.2.8: Использование команды set disassembly-flavor intel . .	15
2.14	Рис 2.2.9: Включение режима псевдографики	16
2.15	Рис 2.3.1: Проверка точки останова	17
2.16	Рис 2.3.2: Установка точки останова и просмотр информации о всех точках останова	18
2.17	Рис 2.4.1: Измененный вид gdb после stepi	20
2.18	Рис 2.4.2: Просмотр регистров с помощью info registers	22
2.19	Рис 2.4.3: Просмотр msg1	23
2.20	Рис 2.4.4: Просмотр msg2	23
2.21	Рис 2.4.5: Изменение первого символа msg1	23
2.22	Рис 2.4.6: Изменение первого символа msg2	24
2.23	Рис 2.4.7: Вывод значения регистра в разных представлениях . .	24
2.24	Рис 2.4.8: Изменение значений регистра ebx	24
2.25	Рис 2.5.1: Копирование файла .asm	25
2.26	Рис 2.5.2: Создание испол. файла	25
2.27	Рис 2.5.3: Загрузка испол. файла с указанием аргументов	26
2.28	Рис 2.5.4: Установка точки останова	26
2.29	Рис 2.5.5: Просмотр адреса вершины стека и позиции	26
2.30	Рис 2.5.6: Просмотр остальных позиций стека	27
3.1	Рис 3.1.1: Демонстрация измененной программы для задания . .	29
3.2	Рис 3.1.2: Проверка программы	30
3.3	Рис 3.2.1: Создание файла	30
3.4	Рис 3.2.2: Демонстрация программы	31
3.5	Рис 3.2.3: Проверка программы	31

3.6	Рис 3.2.4: Получение исп. файла для gdb	32
3.7	Рис 3.2.5: Наблюдение	32
3.8	Рис 3.2.6: Демонстрация измененной программы	33
3.9	Рис 3.2.7: Проверка программы	33

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

2.1 Реализация подпрограмм в NASM

Создадим каталог для программ лабораторной работы № 9, перейдем в него и создадим файл *lab9-1.asm*:

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

Рис. 2.1: Рис 2.1.1: Создание каталога и файла .asm

В качестве примера рассмотрим программу вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы **_calcul**. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучим текст программы (Листинг 9.1).

Введем в файл *lab9-1.asm* текст программы из листинга 9.1.

```

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 ;-----
13 ; Основная программа
14 ;-----
15
16 mov eax, msg
17 call sprint
18 mov ecx, x
19 mov edx, 80
20 call sread
21 mov eax,x
22 call atoi
23 call _calcul ; Вызов подпрограммы _calcul
24 mov eax,result
25 call sprint
26 mov eax,[res]
27 call iprintLF
28 call quit
29
30 ;-----
31 ; Подпрограмма вычисления
32 ; выражения "2x+7"
33
34 _calcul:
35 mov ebx,2
36 mul ebx
37 add eax,7
38 mov [res],eax
39 ret ; выход из подпрограммы

```

Рис. 2.2: Рис 2.1.2: Демонстрация текста программы в файле

Создадим исполняемый файл и проверим его. Результат работы данной программы будет следующим:

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
2x+7=13
```

Рис. 2.3: Рис 2.1.3: Создание файла и его проверка

Изменим текст программы, добавив 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.

```

call sprintf
mov eax,x
call atoi
call _subcalcul ; Вызов подпрограммы _calcul
call _calcul
mov eax,result
call sprintf
mov eax,[res]
call iprintLF
call quit

;-----
; Подпрограмма вычисления
; выражения "2x+7"

_calcul:
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы

_subcalcul:
mov ebx,3
mul ebx
add eax,-1
ret : выход из подпрограммы

```

Рис. 2.4: Рис 2.1.4: Демонстрация измененного текста программы

Создадим исполняемый файл и проверим его работу.

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
2x+7=23
```

Рис. 2.5: Рис 2.1.5: Создание файла и его проверка

2.2 Отладка программ с помощью GDB

Создадим файл *lab9-2.asm* в каталоге *~/work/arch-pc/lab09* и введем в него текст программы из листинга 9.2. (Программа печати сообщения Hello world!):

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

Рис. 2.6: Рис 2.2.1: Создание файла .asm

```

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. 2.7: Рис 2.2.2: Демонстрация текста программы

Получаем исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'

```

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

```

Рис. 2.8: Рис 2.2.3: Получение испол. файла с ключом -g

Загрузим исполняемый файл в отладчик gdb:

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ gdb lab09-2
```

Рис. 2.9: Рис 2.2.4: Загрузка испол. файла в отладчик

Проверим работу программы, запустив ее в оболочке GDB с помощью команды `run` (сокращённо `r`):

```
(gdb) run
Starting program: /home/adchekmarev/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 162827) exited normally]
```

Рис. 2.10: Рис 2.2.5: Проверка работы файла

Для более подробного анализа программы установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её

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

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

Рис. 2.11: Рис 2.2.6: Установка брейкпоинта на метку и запуск программы

Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`

```
(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.
```

Рис. 2.12: Рис 2.2.7: Использование команды disassemble

Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`

```

(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.

```

Рис. 2.13: Рис 2.2.8: Использование команды set disassembly-flavor intel

Перечислите различия отображения синтаксиса машинных команд в режимах ATТ и Intel.

Имена регистров в АТТ начинается с “%”, а в Intel используется более привычный нам синтаксис.

Включим режим псевдографики для более удобного анализа программы (рис. 9.2):

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

native process 162856 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) █
```

Рис. 2.14: Рис 2.2.9: Включение режима псевдографики

2.3 Добавление точек останова

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать или как номер строки программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»: На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверим это с помощью команды `info breakpoints` (кратко `i b`):


```
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 163554 In: _start
(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)
```

Рис. 2.15: Рис 2.3.1: Проверка точки останова

Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции (см. рис. 9.3). Определим адрес предпоследней инструкции (`mov ebx,0x0`) и установим точку останова

```
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

native process 163554 In: _start
(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) 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. 2.16: Рис 2.3.2: Установка точки останова и просмотр информации о всех точках останова

2.4 Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных.

Выполним 5 инструкций с помощью команды `stepi` (или `si`) и проследим за

изменением значений регистров. Изменились значения регистров eax, ecx, edx и ebx.

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd110 0xffffd110
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

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

native process 163554 In: _start
(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) 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) stepi

```

Рис. 2.17: Рис 2.4.1: Измененный вид gdb после stepi

Посмотрим содержимое регистров также можно с помощью команды `info registers` (или `i r`).

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd110 0xffffd110
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

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

native process 163554 In: _start
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd110 0xffffd110
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
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 2.18: Рис 2.4.2: Просмотр регистров с помощью info registers

Для отображения содержимого памяти можно использовать команду х, которая

выдаёт содержимое ячейки памяти по указанному адресу. Формат, в котором выводятся данные, можно задать после имени команды через косую черту: x/NFU. С помощью команды x & также можно посмотреть содержимое переменной. Посмотрим значение переменной msg1 по имени

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

```
"Hello, "
```

Рис. 2.19: Рис 2.4.3: Просмотр msg1

Посмотрим значение переменной msg2 по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрим инструкцию mov esx,msg2 которая записывает в регистр esx адрес переменной msg2

```
(gdb) x/1sb 0x804a008
```

```
0x804a008 <msg2>:
```

```
"world!\n\034"
```

Рис. 2.20: Рис 2.4.4: Просмотр msg2

Изменим значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. При этом перед именем регистра ставится префикс \$, а перед адресом нужно указать в фигурных скобках тип данных (размер сохраняемого значения; в качестве типа данных можно использовать типы языка Си). Изменим первый символ переменной msg1

```
(gdb) set {char}&msg1='h'
```

```
(gdb) x/1sb &msg1
```

```
0x804a000 <msg1>: "hello, "
```

```
(gdb) █
```

Рис. 2.21: Рис 2.4.5: Изменение первого символа msg1

Заменяем любой символ во второй переменной msg2.

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

Рис. 2.22: Рис 2.4.6: Изменение первого символа msg2

Чтобы посмотреть значения регистров используется команда `print /F` (перед именем регистра обязательно ставится префикс `$`). Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра `edx`.

```
(gdb) p/x $edx
$1 = 0x0
(gdb) p/t $edx
$2 = 0
(gdb) p/c $edx
$3 = 0 '\000'
```

Рис. 2.23: Рис 2.4.7: Вывод значения регистра в разных представлениях

С помощью команды `set` изменим значение регистра `ebx`:

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

Рис. 2.24: Рис 2.4.8: Изменение значений регистра ebx

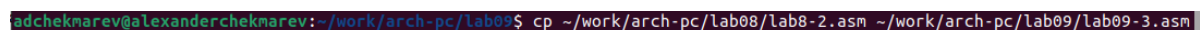
Объясните разницу вывода команд `p/s $ebx`.

В первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется

Завершим выполнение программы с помощью команды `continue` (сокращенно `c`) или `stepi` (сокращенно `si`) и выйдем из GDB с помощью команды `quit` (сокращенно `q`)

2.5 Обработка аргументов командной строки в GDB

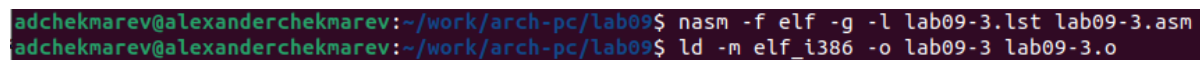
Скопируем файл `lab8-2.asm`, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем `lab09-3.asm`:



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

Рис. 2.25: Рис 2.5.1: Копирование файла `.asm`

Создадим исполняемый файл.



```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Рис. 2.26: Рис 2.5.2: Создание испол. файла

Для загрузки в `gdb` программы с аргументами необходимо использовать ключ `-args`. Загрузим исполняемый файл в отладчик, указав аргументы:

```

adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
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 lab09-3...
(gdb)

```

Рис. 2.27: Рис 2.5.3: Загрузка испол. файла с указанием аргументов

Как отмечалось в предыдущей лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Исследуем расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установим точку останова перед первой инструкцией в программе и запустим ее.

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 11.
(gdb) run
Starting program: /home/adchekmarev/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab09-3.asm:11
11      pop ecx      ; Извлекаем из стека в `ecx` количество
(gdb)

```

Рис. 2.28: Рис 2.5.4: Установка точки останова

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы):

```

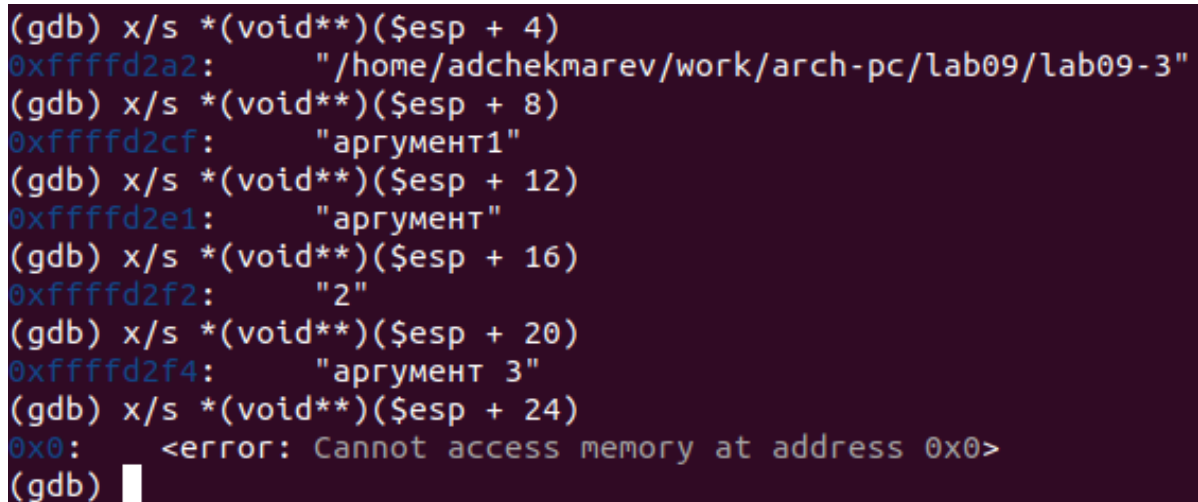
(gdb) x/x $esp
0xffffd0d0:      0x00000005
(gdb)

```

Рис. 2.29: Рис 2.5.5: Просмотр адреса вершины стека и позиции

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

Посмотрим остальные позиции стека – по адресу [esp+4] располагается адрес в памяти где находится имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д.



```
(gdb) x/s *(void**)(esp + 4)
0xffffd2a2:      "/home/adchekmarev/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd2cf:      "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd2e1:      "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd2f2:      "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd2f4:      "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:      <error: Cannot access memory at address 0x0>
(gdb) █
```

Рис. 2.30: Рис 2.5.6: Просмотр остальных позиций стека

Объясните, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.).

Шаг изменения адреса равен 4, так как количество аргументов командной строки равно 4.

3 Самостоятельная работа

Задание №1 Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму.

Скопируем файл *task1.asm* и переместим его в /lab09. Преобразуем этот файл для подпрограммы.

```

1 %include 'in_out.asm'
2
3 SECTION .data
4 msg db "Результат: ",0
5
6 SECTION .text
7 global _start
8 _start:
9
10 pop ecx
11
12 pop edx
13
14 sub ecx,1
15
16 mov esi, 0
17
18 next:
19 cmp ecx,0h
20 jz _end
21
22 pop eax
23 call atoi
24 call calcul
25 loop next
26 _end:
27
28 mov eax, msg
29 call sprint
30 mov eax, esi
31 call iprintLF
32 call quit
33 calcul:
34 mov ebx,8
35 mul ebx
36 sub eax,3
37 add esi,eax
38 ret

```

Рис. 3.1: Рис 3.1.1: Демонстрация измененной программы для задания

Создадим испол. файл и проверим программу с несколькими значениями x

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ nasm -f elf task1.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ ld -m elf_i386 -o task1 task1.o
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ ./task1
Результат: 0
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ ./task1 1 2 3 4
Результат: 68
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ ./task1 3 3 3 4
Результат: 92
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ ./task1 142 1234 1 54
Результат: 11436
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$
```

Рис. 3.2: Рис 3.1.2: Проверка программы

Программа работает корректно

Задание№2 В листинге 9.3 приведена программа вычисления выражения $(3 + 2) \cdot 4 + 5$. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.*

Создадим новый файл *task2.asm*

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

Рис. 3.3: Рис 3.2.1: Создание файла

Напишем программу из Листинга 9.3

```

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. 3.4: Рис 3.2.2: Демонстрация программы

Проверим программу для выявления ошибки

```

adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ nasm -f elf task2.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ ld -m elf_i386 -o task2 task2.o
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab09$ ./task2
Результат: 10

```

Рис. 3.5: Рис 3.2.3: Проверка программы

Правильный ответ 25, в нашем же случае выводится неправильный ответ 10
Получим исполняемый файл для работы с GDB.

```

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

```

Рис. 3.6: Рис 3.2.4: Получение исп. файла для gdb

Запустим его и проставим брейкпоинты для каждой инструкции, связанной с вычислениями. С помощью команды continue продвемся по каждому брейкпоинту, следя за значениями в регистре

```

adchekmarev@alexanderchekmarev: ~/work/arch-pc/lab09
eax      0x804a000      134520832      ecx      0x4           4
eax      0x8           8              ecx      0x4           4
esp      0xffffd110    0xffffd110     ebx      0xa          10
esi      0x0           0              edi      0xa          10
ebp      0x8049105     0x8049105 < start+29>  edi      0x0          0
cs       0x23         35             eflags   0x206         [ PF IF ]
ds       0x2b         43             es       0x2b         43
fs       0x0           0              gs       0x0          0

B+ 0x80490f4 <_start+12> mov    ecx,0x4
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
b+ 0x8049100 <_start+24> mov    eax,0x804a000
b+ 0x8049105 <_start+29> call   0x804900f <sprint>
0x804910a <_start+34> mov    eax,edi
native process 165546 In: _start
Breakpoint 165557 In: _start2.asm:14
Breakpoint 45, _start () at task2.asm:11
(gdb) c
Continuing.
Breakpoint 46, _start () at task2.asm:12
(gdb) c
Continuing.
Breakpoint 47, _start () at task2.asm:13
(gdb) c
Continuing.
Breakpoint 48, _start () at task2.asm:14
(gdb)

```

Рис. 3.7: Рис 3.2.5: Наблюдение

Ошибка была найдена. Исправляем её, добавляя после add ebx,eax mov eax,ebx и заменяя ebx на eax в инструкциях add ebx,5 и mov edi,ebx


```

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

```

Рис. 3.8: Рис 3.2.6: Демонстрация измененной программы

Убедимся, что ошибка исправлена

```

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

```

Рис. 3.9: Рис 3.2.7: Проверка программы

4 Выводы

Я приобрел навыки написания программ с использованием подпрограмм. Познакомился с методами отладки при помощи GDB и его основными возможностями.