

Отчёт по лабораторной работе №9
Простейший вариант

Метвалли Ахмед Фарг Набеев

Содержание

1 Цель работы 5

2 Выполнение лабораторной работы 6

3 Задания для самостоятельной работы 13

4 Выводы 15

Список иллюстраций

2.1

2.2 $f(g(x))$

2.3

2.4

2.5

2.6

2.7

2.8

2.9

2.10

2.11

2.12

2.13

2.14

2.15

2.16

2.17

2.18

2.19

3.1

3.2

3.3

3.4

3.5

3.6

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

Создадим рабочую директорию и файл. Запишем туда программу из листинга,

```
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 2.1: напишем программу, имитирующую сложную функцию. Функции назовем `_calul` и `subcalcul`(рис. 2.2)

```
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$
```

```
ahmad-farg@ahmadfarg-VirtualBox:~$ mkdir ~/work/arch-pc/lab09
ahmad-farg@ahmadfarg-VirtualBox:~$
ahmad-farg@ahmadfarg-VirtualBox:~$ cd ~/work/arch-pc/lab09
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ touch lab09-1.asm
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 2.2: $f(g(x))$

Проверим ее работу (рис. [-fig. 2.3)

```
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2(3x-1)+7=17
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$
```

```

#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:
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
_calcul:
mov ebx, 2
mul ebx
add eax, 7
102 Демидова А. В.
Архитектура ЭВМ
mov [res], eax
ret

```

Рис. 2.3: -

Создадим файл lab10-2.asm и посмотрим, как она работает. Так же проасsembлируем его с другими ключами, чтобы была возможность открыть этот файл через gdb. (рис. 2.4)

```

#include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2(3x-1)+7=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret
_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret

```

```

ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
ahmad-farg@ahmadfarg-VirtualBox:~/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. 2.4: -

Откроем lab10-2 с помощью gdb. Запустим ее там (рис. 2.5)

```

ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ touch lab09-2.asm

```

Рис. 2.5: -

Поставим точку останова(breakpoint) на метке _start. Посмотрим дизассемблированный код, начиная с этой метки. (рис. 2.6)


```

(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. 2.6: -

Так же посмотрим, как выглядит дизассемблированный код с синтаксисом Intel (рис. 2.7)

```

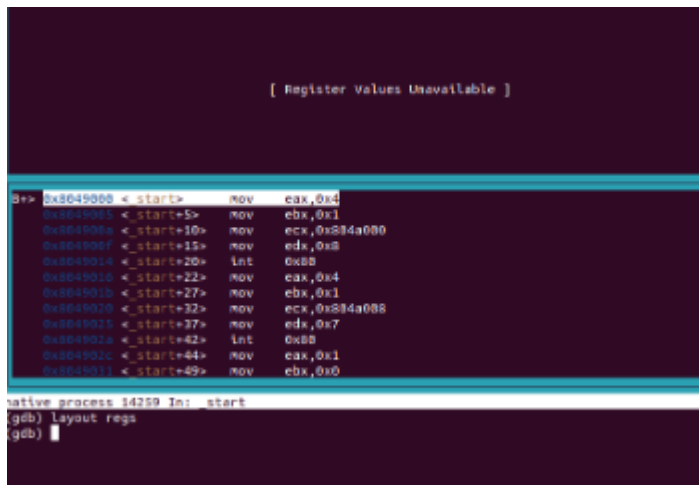
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     eax,0x4
    0x08049005 <+5>:      mov     ebx,0x1
    0x0804900a <+10>:     mov     ecx,0x804a000
    0x0804900f <+15>:     mov     edx,0x8
    0x08049014 <+20>:     int     0x80
    0x08049016 <+22>:     mov     eax,0x4
    0x0804901b <+27>:     mov     ebx,0x1
    0x08049020 <+32>:     mov     ecx,0x804a008
    0x08049025 <+37>:     mov     edx,0x7
    0x0804902a <+42>:     int     0x80
    0x0804902c <+44>:     mov     eax,0x1
    0x08049031 <+49>:     mov     ebx,0x0
    0x08049036 <+54>:     int     0x80
End of assembler dump.
(gdb) █

```

Рис. 2.7: -

В представлении АТТ в виде 16-ричного числа записаны первые аргументы всех команд, а в представлении intel так

записываются адреса вторых аргументов. Включим режим псевдографики, с помощью которого отображается код программы и содержимое регистров(рис. 2.8)



The screenshot shows the GDB interface with the assembly code of the 'start' function. The registers are listed at the bottom, showing their current values.

```

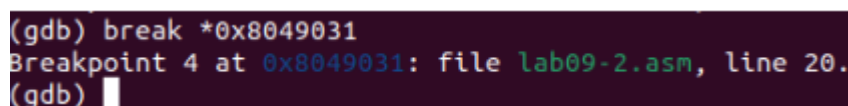
[ Register Values Unavailable ]

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
0x804901c <_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

native process 34259 In: start
(gdb) layout regs
(gdb)
  
```

Рис. 2.8: -

Посмотрим информацию о наших точках останова. Сделать это можно коротко командой `i b` (рис. 2.9)



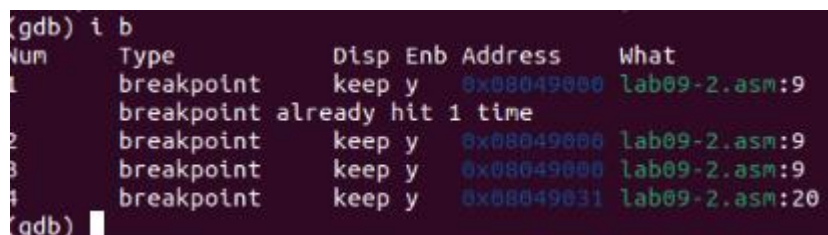
The screenshot shows the GDB interface with the command `i b` executed, displaying the list of breakpoints.

```

(gdb) break *0x8049031
Breakpoint 4 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
  
```

Рис. 2.9: -

добавим еще одну точку останова, но сделаем это по адресу (рис. 2.10)



The screenshot shows the GDB interface with the command `i b` executed, displaying the list of breakpoints.

```

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

Рис. 2.10: -

Так же можно выводить значения регистров. Делается это командой `i r`. Псевдографика представлена на (рис. 2.11)

```

Register group: general
eax      0x8      8      ecx      0x804a000
edx      0x8      8      ebx      0x1
esp      0xffffd170 0xffffd170  ebp      0x0
esi      0x8      8      edi      0x0
eip      0x8049016 0x8049016 < start+22>  eflags  0x202
cs       0x23      35      ss       0x2b
ds       0x2b      43      es       0x2b
fs       0x0      0      gs       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
0x804901b <_start+27>   mov     ebx,0x1
0x8049020 <_start+32>   mov     ecx,0x804a000
0x8049025 <_start+37>   mov     edx,0x7
Help 0x804902a <_start+42> int     0x80
0x804902f <_start+47>   mov     eax,0x1
B+ 0x8049031 <_start+49> mov     ebx,0x0

native process 14259 In: start
(gdb) i b
run
Type      Disp Enb Address  What
breakpoint keep y 0x8049000 lab09-2.asm:19
breakpoint already hit 1 time
breakpoint keep y 0x8049000 lab09-2.asm:9
breakpoint keep y 0x8049000 lab09-2.asm:9
breakpoint keep y 0x8049011 lab09-2.asm:20
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.11: -

В отладчике можно вывести текущее значение переменных. Сделать это можно, например по имени (рис. 2.12) или по адресу (рис. 2.13)

```

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

```

Рис. 2.12: -

```

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

```

Рис. 2.13: -

Так же отладчик позволяет менять значения переменных прямо во время выполнения программы (рис. 2.14)

Рис. 2.14: -

Здесь тоже можно обращаться по адресам переменных (рис. 2.15). здесь был заменен первый символ переменной msg2 на символ отступа.

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

Рис. 2.15: -

Скопируем файл из лабораторной 9, переименуем и создадим исполняемый файл. Откроем отладчик и зададим аргументы. Создадим точку останова на метке `_start` и запустим программу (рис. 2.16)

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

Рис. 2.16: -

Посмотрим на содержимое того, что расположено по адресу, находящемуся в регистре `esp` (рис. 2.17)

```
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
~/work/arch-pc/lab09$ gdb --args lab09-3 2 3 '5'
```

Рис. 2.17: -

Далее посмотрим на все остальные аргументы в стеке. Их адреса располагаются в 4 байтах друг от друга (именно столько занимает элемент стека) (рис. 2.19)

```

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

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xffffd160:    0x00000004
(gdb) █

```

Рис. 2.19: -

```

(gdb) x/x $esp
0xffffd160:    0x00000004
(gdb) x/s *(void**)(esp + 4)
0xffffd336:    "/home/hamdi/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd35d:    "2"
(gdb) x/s *(void**)(esp + 12)
0xffffd35f:    "3"
(gdb) x/s *(void**)(esp + 16)
0xffffd361:    "5"
(gdb) x/s *(void**)(esp + 20)
0x0:    <error: Cannot access memory at address 0x0>

```

3. Задания для самостоятельной работы

Программа из лабораторной 9, но с использованием подпрограмм (рис. 3.1)

```

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. 3.1: -

и проверка ее работоспособности (рис. 3.2)

```
ahmad-farggah@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
```

Рис. 3.2: -

Просмотр регистров, для поиска ошибки в программе из листинга 10.3 (рис. 3.3) и (рис. 3.4)

```
(gdb) c
Continuing.
world!

Breakpoint 4, _start () at lab09-2.asm:20
(gdb)
```

Рис. 3.3: -

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

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

Рис. 3.4: -

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

Ошибка была в строчках

add ebx, eax

mov ecx, 4

mul ecx

add ebx, 5

mov edi, ebx

правильно работающая программа представлена на (рис. 3.5)

```

#include 'in_out.asm'
SECTION .data
div: DB 'result: ',0
SECTION .text
GLOBAL _start
_start:
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add eax,5
mov edi,eax
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 3.5: - Проверка корректности работы программы, после исправлений (рис. 3.6)

```

ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ touch lab09-5.asm
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$

ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$ ./lab09-5
result: 13
ahmad-farg@ahmadfarg-VirtualBox:~/work/arch-pc/lab09$

```

Рис. 3.6: -

4. Выводы

В результате выполнения работы я научился организовывать код в подпрограммы и познакомился с базовыми функциями отладчика gdb.