

Шаблон отчёта по лабораторной работе

Простейший вариант

Дмитрий Сергеевич Кулябов

Содержание

1	1 Цель работы	5
2	2 Задание	6
3	3 Выполнение лабораторной работы	7
3.1	1. Реализация подпрограмм в NASM	7
3.2	2. Отладка программ с помощью GDB	9
3.2.1	2. 1. Добавление точек останова	11
3.2.2	2. 2. Работа с данными программы в GDB	11
3.2.3	2. 3. Обработка аргументов командной строки в GDB	14
3.3	3. Задание для самостоятельной работы	15
4	4 Вывод	20
5	5 Источники	21

Список иллюстраций

Список таблиц

1 1 Цель работы

Целью работы является приобретение навыков написания программ с использованием подпрограмм и ознакомление с методами отладки при помощи GDB и его основными возможностями.

2 2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
 - 2.1. Добавление точек останова
 - 2.2. Работа с данными программы в GDB
 - 2.3. Обработка аргументов командной строки в GDB
3. Задание для самостоятельной работы

3 3 Выполнение лабораторной работы

3.1 1. Реализация подпрограмм в NASM

Для начала я создаю каталог для программ лабораторной работы № 9, перехожу

```
[kanechaeva@fedora arch-pc]$ mkdir ~/work/arch-pc/lab
```

в него и создаю файл lab09-1.asm. (рис. [??])

```
[kanechaeva@fedora lab09]$ touch lab09-1.asm
```

Ввожу в файл lab09-1.asm текст программы из листинга 9.1, скопировав заранее

```
lab09-1.asm      [-M--] 21 L:[ 1+ 0  1/ 35] *(21 /
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
```

файл in_out.asm в папку lab09. (рис. [??])

```
[kanechaeva@fedora lab09]$ nasm -f elf
[kanechaeva@fedora lab09]$ ld -m elf
[kanechaeva@fedora lab09]$ ./lab09-1
Введите x: 6
2x+7=19
[kanechaeva@fedora lab09]$
```

Создаю исполняемый файл и проверяю его работу. (рис. [??])

Теперь изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$. ((рис. [??])

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

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

```
[kanechaeva@fedora lab09]$ nasm -f elf
[kanechaeva@fedora lab09]$ ld -m elf
[kanechaeva@fedora lab09]$ ./lab09-1
Введите x: 6
2*(3*x-1)+7=41
[kanechaeva@fedora lab09]$
```

Создаю исполняемый файл и проверяю его работу. (рис. [??])

3.2 2. Отладка программ с помощью GDB

Создаю файл lab09-2.asm в каталоге ~/work/arch-pc/lab09 и ввожу в него текст

```
lab09-2.asm      [-M--]  8 L:[ 1+20 21/ 21] *(2
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
```

программы из листинга 9.2. ((рис. [??])

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды

```
(gdb) run
Starting program: /home/kanechaeva/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 19873) exited normally]
```

run. (рис. [??])

Для более подробного анализа программы устанавливаю брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запускаю её. Затем смотрю дисассимилированный код программы с помощью команды disassemble начиная с метки _start. Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel.

```

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

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

Включая режим псевдографики для более удобного анализа программы. (рис.

```

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 7554 In: _start L9 PC: 0x8049000
 (gdb) layout regs

[??])

3.2.1 2.1. Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверяю это с помощью команды `info breakpoints`. (рис. [??])

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

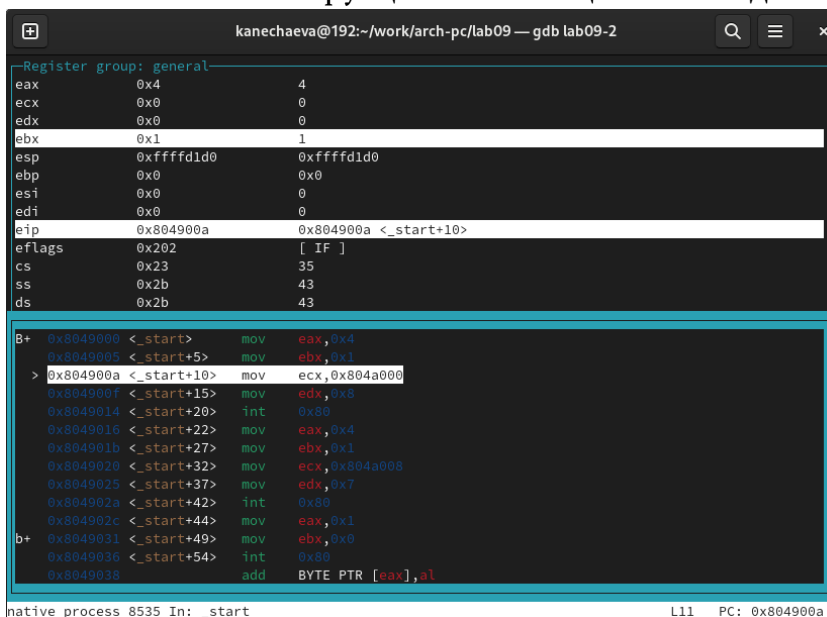
Определяю адрес предпоследней инструкции (`mov ebx,0x0`) и устанавливаю точ-

```
(gdb) break *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
2        breakpoint       keep y   0x08049031 lab09-2.asm:20
(gdb)
```

ку останова. (рис. [??])

3.2.2 2.2. Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды `stepi` (или `si`). (рис. [??])



The screenshot shows the GDB interface with the following content:

Register group: general

Register	Value	Comment
eax	0x4	4
ecx	0x0	0
edx	0x0	0
ebx	0x1	1
esp	0xffffd1d0	0xffffd1d0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x804900a	0x804900a <_start+10>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43

Instruction window:

```
B+ 0x8049000 <_start> mov    eax,0x4
0x8049005 <_start+5> mov    ebx,0x1
> 0x804900a <_start+10> mov    ecx,0x804a000
0x804900f <_start+15> mov    edx,0x5
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
0x8049038 add    BYTE PTR [eax],al
```

native process 8535 In: _start L11 PC: 0x804900a

Изменяются значения

регистров `eax`, `ebx`, `ecx`, `edx`.

Смотрю содержимое регистров с помощью команды `info registers`. (рис. [??])

```

eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

```

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

```

Теперь мне нужно значение переменной msg1 по имени. (рис. [??])

```

(gdb) x/1sb 0x804a000
0x804a008 <msg2>:
(gdb)

```

Затем я смотрю значение переменной msg2 по адресу. (рис. [??])

```

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

```

Изменяю первый символ переменной msg1. (рис. [??])

```

(gdb) set {char}0x804a000='h'
(gdb) x/1sb &msg2
0x804a008 <msg2>:
(gdb)

```

Заменяю третий символ во второй переменной msg2. (рис. [??])

Вывожу в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. (рис. [??])

```
(gdb) p/s $edx
$2 = 8
(gdb) p/t $edx
$3 = 1000
(gdb) p/x $edx
$4 = 0x8
(gdb) 
```

С помощью команды set изменяю значение регистра ebx. (рис. [??])

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

Разница вывода команд

p/s \$ebx в том, что в первый раз программа выдает 50, т.к. это значение символа

2, а во второй раз уже сам символ 2.

Завершаю выполнение программы с помощью команды `continue` (сокращенно

```
(gdb) c
Continuing.
would!

Breakpoint 2, _start () at lab09-2.
(gdb) q
A debugging session is active.

        Inferior 1 [process 8572] w

Quit anyway? (y or n) y
```

с) и выхожу из GDB с помощью команды `quit`. (рис. [??])

3.2.3 2.3. Обработка аргументов командной строки в GDB

Копирую файл `lab8-2.asm`, созданный при выполнении лабораторной работы №8 с программой, выводящей на экран аргументы командной строки, в файл с именем `lab09-3.asm`. После этого создаю исполняемый файл и для загрузки в `gdb` программы с аргументами использую ключ `-args`. Теперь загружаю исполняемый файл в отладчик, указав определенные аргументы. (рис. [??])

```
[kanechaeva@192 lab09]$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3
.asm
[kanechaeva@192 lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[kanechaeva@192 lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[kanechaeva@192 lab09]$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora Linux 13.2-6.fc38
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 lab09-3...
(gdb)
```

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`. Теперь устанавливаю точку останова перед первой инструкцией в программе и

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/kanechaeva/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3
Breakpoint 1, _start () at lab09-3.asm:5
5      pop есх ; Извлекаем из стека в `есх` количество
(gdb)
```

запускаю ее. (рис. [??])

Адрес вершины стека хранится в регистре есп и по адресу вершины стека я смотрю количество аргументов командной строки (включая имя программы).(рис.

```
(gdb) x/x $esp
0xffffd180:      0x00000005
(gdb)
```

[??])

Как видно, число аргументов равно 5 – это имя программы lab09-3 и аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Теперь я смотрю остальные позиции стека: [esp+4] – адрес с именем программы, [esp+8] - адрес первого аргумента, [esp+12] –

```
(gdb) x/s *(void**)( $esp + 4)
0xffffd32f:      "/home/kanechaeva/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xffffd35b:      "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xffffd36d:      "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xffffd37e:      "2"
(gdb) x/s *(void**)( $esp + 20)
0xffffd380:      "аргумент 3"
(gdb) x/s *(void**)( $esp + 24)
0x0:      <error: Cannot access memory at address 0x0>
(gdb)
```

второго и т.д. (рис. [??])

Шаг изменения адреса равен 4, т.к. размер ячейки памяти со значением, на которую указывает есп увеличивается на 4.

3.3 3.Задание для самостоятельной работы

1. Требуется преобразовать программу из задания для самостоятельной работы лабораторной работы №8, реализовав вычисление значения функции $\boxtimes(\boxtimes)$ как подпрограмму. Для начала копирую файл lab8-5.asm в lab09 с именем lab9-4.asm и уже там начинаю работу над кодом. (рис. [??])

```

lab9-4.asm      [-M--] 21 L: [ 1+ 0  1/ 33] *(21 / 338b) 0010
%include 'in_out.asm'
SECTION .data
result db 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0

next:
cmp ecx,0h
jz _end
pop eax
call atoi
call _calcul
add esi,eax
loop next

_end:
mov eax, result
call sprint
mov eax,esi
call iprintLF
call quit

_calcul:
mov ebx,15
mul ebx
add eax,-9
ret

```

```

[kanechaeva@192 lab09]$ nasm -f elf lab9-4.a
[kanechaeva@192 lab09]$ ld -m elf_i386 -o lab9-4
[kanechaeva@192 lab09]$ ./lab9-4 1 2 3 4
Результат: 114
[kanechaeva@192 lab09]$

```

Создаю исполняемый файл и запускаю его. [??]

2. В листинге 9.3 приведена программа вычисления выражения $(3 + 2)$
☒ $4 + 5$. При запуске данная программа дает неверный результат. [??]


```

[kanechaeva@192 lab09]$ nasm -f elf -g -l lab9-5.lst lab9-5.asm
[kanechaeva@192 lab09]$ ld -m elf_i386 -o lab9-5 lab9-5.o
[kanechaeva@192 lab09]$ gdb lab9-5
GNU gdb (GDB) Fedora Linux 13.2-6.fc38
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 lab9-5...
(gdb) r
Starting program: /home/kanechaeva/work/arch-pc/lab09/lab9-5

This GDB supports auto-downloading debuginfo from the following URL
<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
.Результат: 10

```

Включаю режим псевдографики для более удобного анализа программы. [??])

```

eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490fb 0x80490fb <_start+19>

B+ 0x80490e8 <_start>    mov    $0x3,%ebx
0x80490ed <_start+5>    mov    $0x2,%eax
0x80490f2 <_start+10>   add    %eax,%ebx
0x80490f4 <_start+12>   mov    $0x4,%ecx
0x80490f9 <_start+17>   mul    %ecx
> 0x80490fb <_start+19> add    $0x5,%ebx
0x80490fe <_start+22>   mov    %ebx,%edi
0x8049100 <_start+24>   mov    $0x804a000,%eax
0x8049105 <_start+29>   call   0x804900f <sprint>
0x804910a <_start+34>   mov    %edi,%eax

native process 13477 In: _start
Start it from the beginning? (y or n) yStarting program: /home
lab09/lab9-5

Breakpoint 1, _start () at lab9-5.asm:8
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si

```

Сначала в ebx записывается 3, в eax записывается 2, потом они складываются и в переменной ebx, в ecx записывается 4. После этого идет mul ecx и регистр eax становится 8. Это произошло, т.к. по умолчанию идет умножение на то что записано в eax, а там 2, а $2*4=8$. Следовательно, надо записать результат сложения в переменную eax. И во всех дальнейших арифметических действиях я также

```

lab9-5.asm      [----] 10 L:[ 1+13
#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

```

заменяю edx на eax. [??])

```

[kanechaeva@192 lab09]$ nasm -f elf lab9-5.asm
[kanechaeva@192 lab09]$ ld -m elf_i386 -o lab9
[kanechaeva@192 lab09]$ ./lab9-5
Результат: 25
[kanechaeva@192 lab09]$

```

Исправляю программу и запускаю ее снова. [??])

4 4 Вывод

При выполнении данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.

5 5 Источники

1. ТУИС – Архитектура ЭВМ – [Электронный ресурс] - <https://esystem.rudn.ru/mod/resource/view.php?id=7089>