

Отчёт по лабораторной работе №9

Архитектура компьютера

Иванов Сергей Владимирович

Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Задание для самостоятельной работы.	14
4	Выводы	18

Список иллюстраций

2.1	Файл lab09-1.asm	5
2.2	Работа файла lab9-1.asm	5
2.3	Изменение файла	6
2.4	Результат работы файла lab09-1.asm	6
2.5	Запуск lab9-2.asm в gdb	7
2.6	Брейкпоинт	7
2.7	Дисассимилированный код	8
2.8	Режим псевдографики	9
2.9	info breakpoints	9
2.10	Команда stepi	10
2.11	Изменение символа	10
2.12	Изменение символа в msg2	10
2.13	Регистр edx	11
2.14	Значения регистра ebx	11
2.15	Файл lab9-3.asm	12
2.16	Адрес вершины стека	12
2.17	Все вершины стека	12
3.1	Программа lab9-4	16
3.2	Поиск ошибок с помощью отладчика	16
3.3	Запуск файла	17

1 Цель работы

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

2 Выполнение лабораторной работы

Создадим каталог для программ лабораторной работы № 9, перейдем в него и создадим файл lab09-1.asm.(рис. 2.1).

```
svivanov1@svivanov1:~$ mkdir ~/work/arch-pc/lab09
svivanov1@svivanov1:~$ cd ~/work/arch-pc/lab09
svivanov1@svivanov1:~/work/arch-pc/lab09$ touch lab9-1.asm
svivanov1@svivanov1:~/work/arch-pc/lab09$ mc
```

Рис. 2.1: Файл lab09-1.asm

Введем в файл программу листинга 9.1, создадим исполняемый файл и проверим его работу.

```
svivanov1@svivanov1:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
svivanov1@svivanov1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
svivanov1@svivanov1:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 1
2x+7=9
svivanov1@svivanov1:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 2
2x+7=11
```

Рис. 2.2: Работа файла lab9-1.asm

Изменим текст программы добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. (рис. 2.3).

```

; Подпрограмма вычисления выражения "2x + 7"
_calcul:
    ; Вычисление g(x) в подпрограмме _subcalcul
    call _subcalcul
    ; Результат g(x) возвращается в eax
    ; Теперь вычисляем f(g(x)) = 2 * g(x) + 7
    mov ebx, 2
    imul eax, ebx
    add eax, 7
    ; Результат f(g(x)) сохраняется в [res]
    mov [res], eax
    ret ; выход из подпрограммы

; Подпрограмма вычисления выражения "3x - 1"
_subcalcul:
    mov ebx, 3
    imul eax, ebx
    sub eax, 1
    ret ; возвращаем результат в eax

```

Рис. 2.3: Изменение файла

Создадим исполняемый файл и запустим его, программа работает корректно. (рис. 2.4).

```

svivanov1@svivanov1:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
svivanov1@svivanov1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
svivanov1@svivanov1:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 2
f(g(x))=17

```

Рис. 2.4: Результат работы файла lab09-1.asm

Создадим файл lab09-2.asm с текстом программы из Листинга 9.2. Получим исполняемый файл. Загрузим исполняемый файл в отладчик gdb. Проверим работу программы, запустив ее в оболочке GDB с помощью команды run(рис. 2.5).

```

svivanov1@svivanov1:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
svivanov1@svivanov1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
svivanov1@svivanov1:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Ubuntu 13.1-2ubuntu2) 13.1
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-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 lab9-2...
(gdb) run
Starting program: /home/svivanov1/work/arch-pc/lab09/lab9-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Hello, world!
[Inferior 1 (process 4073) exited normally]
(gdb)

```

Рис. 2.5: Запуск lab9-2.asm в gdb

Установим брейкпоинт на метку `_start`, и запустим её(рис. 2.6).

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/svivanov1/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4
(gdb)

```

Рис. 2.6: Брейкпоинт

Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`. Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`(рис. 2.7).

```

(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. 2.7: Дисассимилированный код

Различия отображения синтаксиса машинных команд в режимах ATТ и Intel: противоположное расположение операнда-источника и операнда-приемника; в ATТ регистры пишутся после ‘%’, а непосредственные операнды после ‘\$’, в синтаксисе Intel операнды никак не помечаются.

Включим режим псевдографики для более удобного анализа программы(рис. 2.8).


```

[ 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
0x8049016 <_start+22>  mov    eax,0x4
0x804901b <_start+27>  mov    ebx,0x1
0x8049020 <_start+32>  mov    ecx,0x804a008
0x8049025 <_start+37>  mov    edx,0x7

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

```

Рис. 2.8: Режим псевдографики

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверим это с помощью команды `info breakpoints`. Установим еще одну точку останова по адресу предпоследней инструкции. Посмотрим информацию о всех установленных точках останова (рис. 2.9).

```

[ Register Values Unavailable ]

B+> 0x8049000 <_start>    mov    $0x4,%eax
0x8049005 <_start+5>   mov    $0x1,%ebx
0x804900a <_start+10>  mov    $0x804a000,%ecx
0x804900f <_start+15>  mov    $0x8,%edx
0x8049014 <_start+20>  int     $0x80
0x8049016 <_start+22>  mov    $0x4,%eax

native process 4127 In: _start L9 PC: 0x8049000
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x08049000 lab9-2.asm:9
      breakpoint already hit 1 time
2     breakpoint       keep y  0x08049031 lab9-2.asm:20
(gdb)

```

Рис. 2.9: info breakpoints

Выполним 5 инструкций с помощью команды `stepi` и проследим за изменением значений регистров. В результате изменяются значения регистров `eax`, `ebx`, `ecx`,

edx(рис. 2.10).

```
0x804900a <_start+10> mov $0x804a000,%ecx
0x804900f <_start+15> mov $0x8,%edx
0x8049014 <_start+20> int $0x80
> 0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27> mov $0x1,%ebx
0x8049020 <_start+32> mov $0x804a008,%ecx

native process 4127 In: _start L14 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xfffffd100 0xfffffd100
ebp      0x0      0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 2.10: Команда stepi

Посмотрите значение переменной msg1 и msg2 по имени. Изменим первый символ переменной msg1(рис. 2.11).

```
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb &msg2
0x804a008 <msg2>: "world!\n\034"
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) █
```

Рис. 2.11: Изменение символа

Заменяем символ во второй переменной msg2.(рис. 2.12).

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

Рис. 2.12: Изменение символа в msg2

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. (рис. 2.13).

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

Рис. 2.13: Регистр edx

С помощью команды set изменим значение регистра ebx: (рис. 2.14).

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

Рис. 2.14: Значения регистра ebx

Используя команду set изменили значение регистра ebx сначала на символ '2', а затем на число 2, и сравнили вывод значения регистра в десятичном формате. В результате присвоения регистра значение символа '2', выводится число 50, что соответствует символу в '2' в таблице ASCII

Скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки в файл с именем lab9-3.asm. Создадим исполняемый файл. Загрузим исполняемый файл в отладчик, указав аргументы: (рис. 2.15).

```

svivanov1@svivanov1:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
svivanov1@svivanov1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
svivanov1@svivanov1:~/work/arch-pc/lab09$ gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Ubuntu 13.1-2ubuntu2) 13.1
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-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 lab9-3...
(gdb)

```

Рис. 2.15: Файл lab9-3.asm

Для начала установим точку останова перед первой инструкцией в программе и запустим ее. (рис. 2.16).

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/svivanov1/work/arch-pc/lab09/lab9-3 аргумент1 аргумент 2 аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.

Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)

```

Рис. 2.16: Адрес вершины стека

Посмотрим остальные позиции стека. (рис. 2.17).

```

(gdb) x/s *(void**)(esp + 4)
0xffffd2bb:  "/home/svivanov1/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd2e5:  "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd2f7:  "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd308:  "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd30a:  "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:  <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 2.17: Все вершины стека

В первом хранится адрес, в остальных хранятся элементы. Элементы расположены с интервалом в 4 единицы, так как стек может хранить до 4 байт:каждый элемент стека занимает 4 байта, поэтому для получения следующего элемента стека мы добавляем 4 к адресу вершины.

3 Задание для самостоятельной работы.

- 1) Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. Запустим файл. Программа работает корректно. (рис. 3.1).

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg db "Результат: ", 0
```

```
msg1 db "Функция:  $f(x)=7+2x$ ", 0
```

```
SECTION .text
```

```
global _start
```

```
; Подпрограмма для вычисления функции  $f(x)$ 
```

```
calculate_f:
```

```
; Вход: eax - значение x
```

```
; Выход: eax - результат функции  $f(x)$ 
```

```
add eax, eax ; умножение x на 2
```

```
add eax, 7 ; добавление 7
```

```
ret
```

```
_start:
```

```

mov eax, msg1
call sprintf
pop ecx ; Извлекаем из стека в ecx количество аргументов
pop edx ; Извлекаем из стека в edx имя программы

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

pop eax ; Иначе извлекаем следующий аргумент из стека
call atoi ; Преобразуем символ в число
call calculate_f ; Вызываем подпрограмму для вычисления f(x)
add esi, eax ; Добавляем к промежуточной сумме следующий аргумент (esi = esi + eax)
loop next ; Переход к обработке следующего аргумента

```

_end:

```

mov eax, msg ; Вывод сообщения "Результат: "
call sprintf
mov eax, esi ; Записываем сумму в регистр eax
call sprintf ; Печать результата
call quit ; Завершение программы

```

```
svivanov1@svivanov1:~/work/arch-pc/lab09$ ./lab9-4 1 2 3
Функция: f(x)=7+2x
Результат: 33
svivanov1@svivanov1:~/work/arch-pc/lab09$ ./lab9-4 11 12
Функция: f(x)=7+2x
Результат: 60
svivanov1@svivanov1:~/work/arch-pc/lab09$
```

Рис. 3.1: Программа lab9-4

- 2) Создадим файл с текстом из листинга 9.3, где приведена программа вычисления выражения $(3+2)*4+5$. При запуске данная программа дает неверный результат

Проанализируем с помощью отладчика GDB изменение значений регистров, чтобы найти ошибку. Установим брейкпоинт `b _start`, включим режим псевдографики и начнём поочередно выполнять команды и следить за значениями регистров. Ошибка в том, что результат $3+2=5$ записан в регистре `ebx` (команда `add ebx, eax`), но команда `mul` всегда перемножает значение регистра `eax` с указанным сомножителем, поэтому `mul ecx` дает неверный результат. Затем, к нему добавляется число 5, результат запоминается в регистре `edi`, а затем выводится (рис. 3.2).

eax	0x8	8
ecx	0x4	4
edx	0x0	0
ebx	0x5	5
esp	0xffffd100	0xffffd100
ebp	0x0	0x0


```

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 add    %al,(%eax)

```

native process 3048 In: _start L13 PC: 0x80490

```

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

```

Рис. 3.2: Поиск ошибок с помощью отладчика

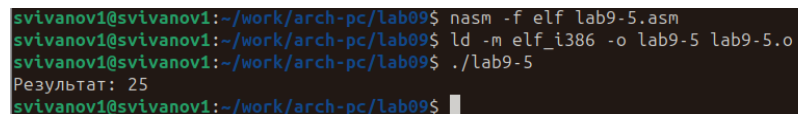
Исправим текст программы:

```
%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
```

Запустим файл. Программа работает корректно.(рис. 3.3).



```
svivanov1@svivanov1:~/work/arch-pc/lab09$ nasm -f elf lab9-5.asm
svivanov1@svivanov1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
svivanov1@svivanov1:~/work/arch-pc/lab09$ ./lab9-5
Результат: 25
svivanov1@svivanov1:~/work/arch-pc/lab09$
```

Рис. 3.3: Запуск файла

4 Выводы

В результате выполнения лабораторной работы мы приобрели навыки написания программ с использованием подпрограмм и познакомилась с методами отладки при помощи GDB и его основными возможностями.