

Отчет по лабораторной работе №9

Архитектура компьютера

Дмитрий Константинович Кобзев

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Самостоятельная работа	16
5	Выводы	19
	Список литературы	20

Список иллюстраций

Список таблиц

1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Задание

1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму.
2. В листинге 9.3 приведена программа вычисления выражения $(3 + 2) * 4 + 5$. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

3 Выполнение лабораторной работы I

[1–6]

Создаем каталог для программ лабораторной работы № 9, переходим в него и

```
dkkobzev@dk6n54:~$ mkdir ~/work/arch-pc/lab09
```

```
dkkobzev@dk6n54:~$ cd ~/work/arch-pc/lab09
```

создаем файл lab09-1.asm (рис. 1.1).

```
dkkobzev@dk6n54:~/work/arch-pc/lab09$ touch lab09-1.asm
```

Вводим в файл программу листинга 9.1, создаем исполняемый файл и проверя-

```

%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
call sprint
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
mov eax, msg ; вызов подпрограммы печати сообщения
call sprint ; 'Введите x: '
mov ecx, x
mov edx, 80
call sread ; вызов подпрограммы ввода сообщения
mov eax, x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret

```

ем его работу (рис. 1.2), (рис. 1.3).

```

dkkobzev@dk6n54:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
dkkobzev@dk6n54:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
dkkobzev@dk6n54:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
2x+7=13

```


Изменяем текст программы добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x +$

```
_calcul:
call _subcalcul
mov ebx,2
imul eax,ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы
_subcalcul:
mov ebx,3
imul eax,ebx
sub eax,1
ret
```

7, $g(x) = 3x - 1$ (рис. 1.4).

```
dkkobzev@dk6n54:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
dkkobzev@dk6n54:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
dkkobzev@dk6n54:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 2
f(g(x))=17
```

Создаем исполняемый файл и запускаем его (рис. 1.5).

Создаем файл `lab09-2.asm` с текстом программы из Листинга 9.2. Получаем исполняемый файл. Загружаем исполняемый файл в отладчик `gdb`. Проверяем работу программы, запустив ее в оболочке GDB с помощью команды `run` (рис. 1.6), (рис. 1.7).

```
dkkobzev@dk6n54:~/work/arch-pc/lab09$ touch lab09-2.asm
dkkobzev@dk6n54:~/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) run
Starting program: /home/dkkobzev/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 8573) exited normally]
(gdb)
```

Устанавливаем брейкпоинт на метку `_start`, и запускаем её (рис. 1.8).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/dkkobzev/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
```

Посмотрим дисассимилированный код программы с помощью команды

```
(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.
```

disassemble начиная с метки _start (рис. 1.9).

Переключитесь на отображение команд с Intel'овским синтаксисом, введя ко-

```
(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,0x804
      0x0804900f <+15>:     mov     edx,0x8
      0x08049014 <+20>:     int     0x80
      0x08049016 <+22>:     mov     eax,0x4
      0x0804901b <+27>:     mov     ebx,0x1
      0x08049020 <+32>:     mov     ecx,0x804
      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.
```

манду set disassembly-flavor intel (рис. 1.10).

Различия отображения синтаксиса машинных команд в режимах ATТ и Intel: противоположное расположение операнда-источника и операнда-приемника;

в АТТ регистры пишутся после '%', а непосредственные операнды после '\$', в синтаксисе Intel операнды никак не помечаются.

Включаем режим псевдографики для более удобного анализа про-

```

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
0x8049038 add     BYTE PTR [eax],al
0x804903a add     BYTE PTR [eax],al
0x804903c add     BYTE PTR [eax],al
0x804903e add     BYTE PTR [eax],al
0x8049040 add     BYTE PTR [eax],al
0x8049042 add     BYTE PTR [eax],al
0x8049044 add     BYTE PTR [eax],al
0x8049046 add     BYTE PTR [eax],al
0x8049048 add     BYTE PTR [eax],al
0x804904a add     BYTE PTR [eax],al
0x804904c add     BYTE PTR [eax],al
0x804904e add     BYTE PTR [eax],al
0x8049050 add     BYTE PTR [eax],al
0x8049052 add     BYTE PTR [eax],al
0x8049054 add     BYTE PTR [eax],al
0x8049056 add     BYTE PTR [eax],al
  
```

native process 8632 In: _start L9 PC: 0x8049000

граммы (рис. 1.11).

[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
0x8049038 add     BYTE PTR [eax],al
0x804903a add     BYTE PTR [eax],al
  
```

native process 8632 In: _start L9 PC: 0x8049000

На предыдущих шагах была установлена точка останова по имени метки (_start). Проверяем это с помощью команды `info breakpoints`. Устанавливаем еще одну точку останова по адресу предпоследней инструк-

ции. Смотрим информацию о всех установленных точках останова (рис.

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

1.13), (рис. 1.14).

```
(gdb) break *0x8049031
Breakpoint 7 at 0x8049031: file lab09-2.asm, line 20.
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
2        breakpoint     keep y   0x08049000 lab09-2.asm:9
3        breakpoint     keep y   0x08049000 lab09-2.asm:9
4        breakpoint     keep y   <PENDING> mov ebx , 0x0
5        breakpoint     keep y   <PENDING> mov ebx , 0x0
6        breakpoint     keep y   <PENDING> b *0x8049031
7        breakpoint     keep y   0x08049031 lab09-2.asm:20
```

Выполняем 5 инструкций с помощью команды step и проследим за изменением значений регистров. В результате изменяются значения регистров eax, ebx,

```
Register group: general
eax      0x4          4
ecx      [ Register Values Unavailable ]
edx      0x0          0
ebx      0x0          0
esp      0xffffd270   0xffffd270

B+ 0x8049000 <_start> mov $0x4,%eax
> 0x8049005 <_start+5> mov $0x1,%ebx
0x804900a <_start+10> mov $0x804a000,%ecx
0x8049014 <_start+20> int $0x80
0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27> mov $0x1,%ebx

exec No process in: L?? PC: ??
native process 20673 In: _start L10 PC: 0x8049005
(gdb) run
The program being debugged has been started already.
start it from the beginning? (y or n) yStarting program: /home/dkkobzev/work/arc
n-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
(gdb) si
```

ecx, edx (рис. 1.15).

Смотрим значение переменной msg1 по имени (рис. 1.16).

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

Изменяем первый символ переменной msg1 (рис. 1.17).

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

```
(gdb) x/1sb &msg2
0x804a008 <msg2>:
(gdb) set {char}&msg2
(gdb) x/1sb &msg2
0x804a008 <msg2>:
```

Заменяем любой символ во второй переменной msg2 (рис. 1.18).

Выводим в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. 1.19).

```
(gdb) p/s $edx
$1 = 7
(gdb) p/t $edx
$2 = 111
(gdb) p/x $edx
$3 = 0x7
```

С помощью команды set изменяем значение регистра ebx (рис. 1.20).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx2=2
$5 = 2
```

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

Копируем 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` (рис. 1.21).

```
dkkobzev@dk6n54:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm /arch-pc/lab09/lab09-3.asm
```

Создаем исполняемый файл (рис. 1.22).

```
dkkobzev@dk6n54:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
dkkobzev@dk6n54:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Загружаем исполняемый файл в отладчик, указав аргументы (рис. 1.23).

```
dkkobzev@dk6n54:~/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...
```

Устанавливаем точку останова перед первой инструкцией в программе и за-

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/dkkobzev/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
```

пускаем ее (рис. 1.24).

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

```
(gdb) x/x $esp
0xffffd220: 0x00000005
```

программы (рис. 1.25).

```

(gdb) x/s *(void**)(esp + 4)
0xffffd3df:  "/home/dkkobzev/work/arch-pc/lab0
(gdb) x/1 *(void**)(esp + 8)
0xffffd409:  "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd41b:  "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd42c:  "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd42e:  "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:  <error: Cannot access memory at address 0

```

Смотрим остальные позиции стека (рис. 1.26).

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.

4 Самостоятельная работа

Задание 1. Преобразовываем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму (рис. 2.1), (рис. 2.2), (рис. 2.3).

```
%include 'in_out.asm'
SECTION .data
msg db 'Результат: ',0
msg1 db "Функция: f(x)=7+2x",0
SECTION .text
GLOBAL _start
calculate_f:
add eax, eax
add eax, 7
ret
_start:
mov eax, msg1
call sprintf
pop ecx
pop edx
sub ecx, 1 ; Уменьшаем ecx на 1 (количество аргументов)
mov esi, 0 ; Используем esi для хранения промежуточной суммы
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
add esi, eax ; Добавляем к промежуточной сумме следующий аргумент
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 iprintf ; Печать результата
call quit ; Завершение программы
```

```
dkkobzev@dk6n54:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/sr.asm ~/work/arch-pc/lab09/sr.asm
dkkobzev@dk6n54:~/work/arch-pc/lab09$ nasm -f elf sr.asm
dkkobzev@dk6n54:~/work/arch-pc/lab09$ ld -m elf_i386 -o sr sr.o
dkkobzev@dk6n54:~/work/arch-pc/lab09$ ./sr 1 2 3
Функция: f(x)=7+2x
Результат: 33
```

Задание 2. В листинге 9.3 приведена программа вычисления выражения $(3 + 2) * 4 + 5$. При запуске данная программа дает неверный результат. Проверяем это. С помощью отладчика GDB, анализируя изменения значений регистров, определяем ошибку и исправляем ее (рис. 2.4),


```

%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. 2.5), (рис. 2.6).

```

%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

```

```

dkkobzev@dk6n54:~/work/arch-pc/lab09$ nasm
dkkobzev@dk6n54:~/work/arch-pc/lab09$ ld -m
dkkobzev@dk6n54:~/work/arch-pc/lab09$ ./sr2
Результат: 25

```

5 Выводы

В ходе выполнения лабораторной работы мною были приобретены навыки написания программ с использованием подпрограмм. Также я познакомился с методами отладки при помощи GDB и его основными возможностями.

Список литературы

1. GNU Bash Manual [Электронный ресурс]. Free Software Foundation, 2016.
URL: <https://www.gnu.org/software/bash/manual/>.
2. Newham C. Learning the bash Shell: Unix Shell Programming. O'Reilly Media, 2005. 354 с.
3. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 с.
4. Robbins A. Bash Pocket Reference. O'Reilly Media, 2016. 156 с.
5. Таненбаум Э. Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 874 с.
6. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.