

Отчёт по лабораторной работе №9

Дисциплина: Архитектура Компьютера

Азарцова Вероника Валерьевна

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	7
4	Задания для самостоятельной работы	14
5	Выводы	17
	Список литературы	18

Список иллюстраций

3.1	Создание каталога lab09	7
3.2	lab9-1.asm	7
3.3	Запуск lab9-1.asm	8
3.4	lab9-2.asm	8
3.5	Загрузка lab9-2.asm в GDB	8
3.6	Запуск в оболочке GDB	9
3.7	Брейпоинт на метке _start	9
3.8	Дисассимилированный код программы	9
3.9	Дисассимилированный код программы с синтаксом intel	9
3.10	Режим псевдографики	10
3.11	Проверка меток	10
3.12	Установка второй метки и проверка	10
3.13	5 инструкций с помощью stepi	11
3.14	Просмотр значения переменной	11
3.15	Изменение символов в переменных	11
3.16	Значение регистра edx в разных форматах	12
3.17	Смена значения регистра ebx	12
3.18	Копирование, создание исполняемого файла, загрузка	12
3.19	Запуск с точкой останова	13
3.20	Просмотр позиций стека	13

Список таблиц

1 Цель работы

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

2 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа:

1. Обнаружение ошибки;
2. Поиск её местонахождения;
3. Определение причины ошибки;
4. Исправление ошибки.

Наиболее часто применяют следующие методы отладки:

1. Создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);
2. Использование специальных программ-отладчиков.

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом.

Для вызова подпрограммы из основной программы используется инструкция call, которая заносит адрес следующей инструкции в стек и загружает в регистр еір адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы.

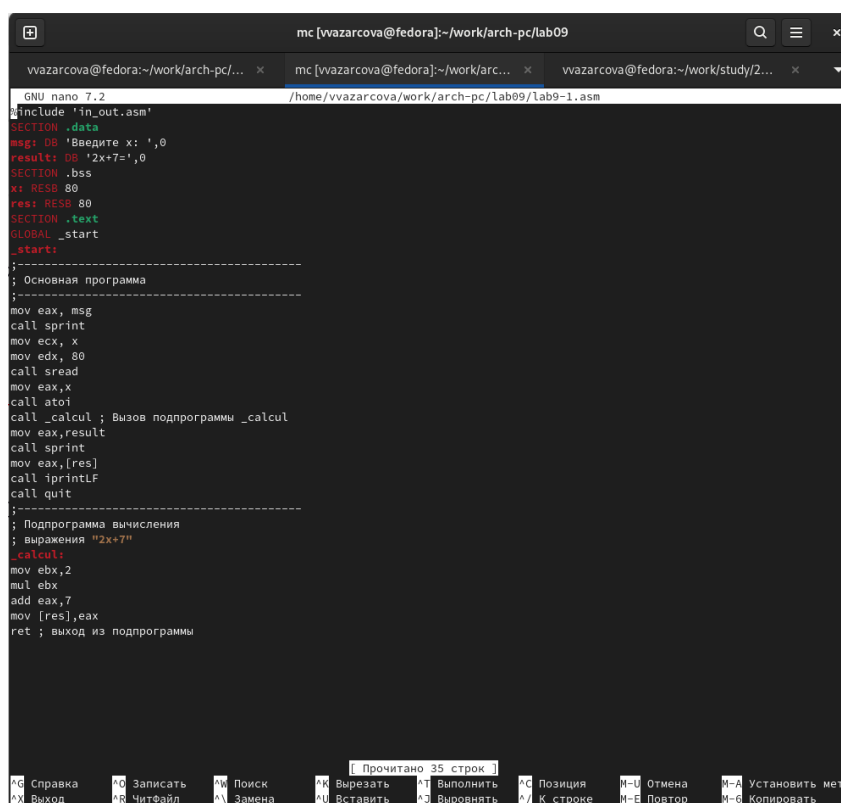
3 Выполнение лабораторной работы

1. Создаю каталог для выполнения лабораторной работы №9 и файл lab09-1.asm в нём (рис. 3.1).

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

Рис. 3.1: Создание каталога lab09

Ввожу текст программы из листинга в файл lab09-1.asm (рис. 3.2).



```
GNU nano 7.2 /home/vvazarcova/work/arch-pc/lab09/lab9-1.asm
#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 iprintf
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы
```

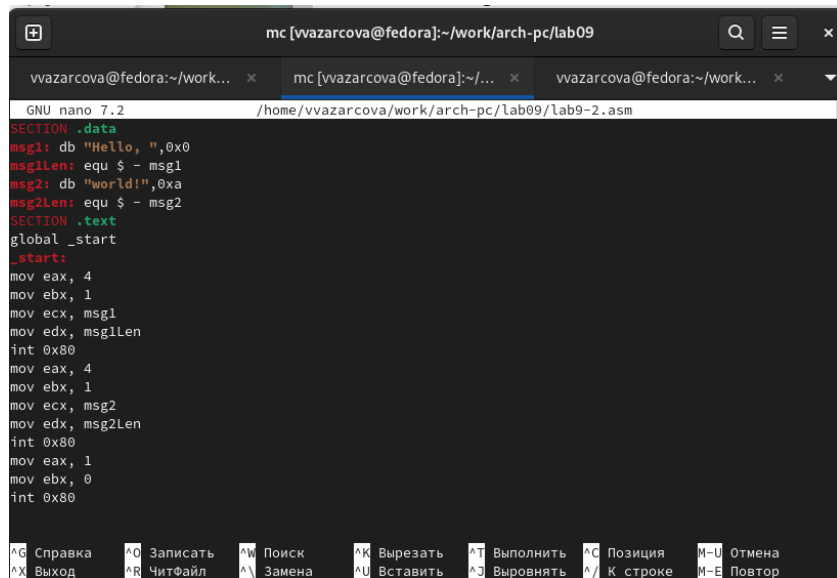
Рис. 3.2: lab9-1.asm

Создаю исполняемый файл и проверяю его работу (рис. 3.3).

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

Рис. 3.3: Запуск lab9-1.asm

2. Создаю файл lab9-2.asm с текстом программа печати сообщения “Hello world!” (рис. 3.4).



```
GNU nano 7.2 /home/vvazarcova/work/arch-pc/lab09/lab9-2.asm
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
^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция ^M-U Отмена
^X Выход ^R ЧитФайл ^A Замена ^U Вставить ^D Выровнять ^/_ К строке ^M-E Повтор
```

Рис. 3.4: lab9-2.asm

Получаю исполняемый файл. Для работы с GDB в исполняемый файл добавляю отладочную информацию, для этого трансляцию программы провожу с ключом ‘-g’ и загружаю исполняемый файл в отладчик gdb (рис. 3.5).

```
vvazarcova@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
vvazarcova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
ld: неизвестный параметр «-0»
ld: используйте --help для получения информации о параметрах
vvazarcova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
vvazarcova@fedora:~/work/arch-pc/lab09$ gdb lab9-2
```

Рис. 3.5: Загрузка lab9-2.asm в GDB

Запускаю программу в оболочке GDB (рис. 3.6).


```
(gdb) run
Starting program: /home/vvazarcova/work/arch-pc/lab09/lab9-2
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Download failed: Нет маршрута до узла. Continuing without separate debug info for system-supplied DSO at 0xf7ffc000.
Hello, world!
[Inferior 1 (process 19460) exited normally]
(gdb)
```

Рис. 3.6: Запуск в оболочке GDB

Запускаю программу с брейпоинтом на метке `_start` (рис. 3.7).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/vvazarcova/work/arch-pc/lab09/lab9-2
Downloading separate debug info for system-supplied DSO at 0xf7ffc000

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

Рис. 3.7: Брейпоинт на метке `_start`

Смотрю на дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 3.8).

```
(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. 3.8: Дисассимилированный код программы

Смотрю на дисассимилированный код программы с синтаксисом intel с помощью команды `disassemble` начиная с метки `_start` (рис. 3.9).

```
(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. 3.9: Дисассимилированный код программы с синтаксисом intel

Эти два отображения различаются тем, что в Intel'овском сначала пишется регистр, а потом адрес, причем без символа доллара и без символа процента, т.е. его вид упрощенный.

Включаю режим псевдографики для более удобного анализа программы (рис. 3.10).

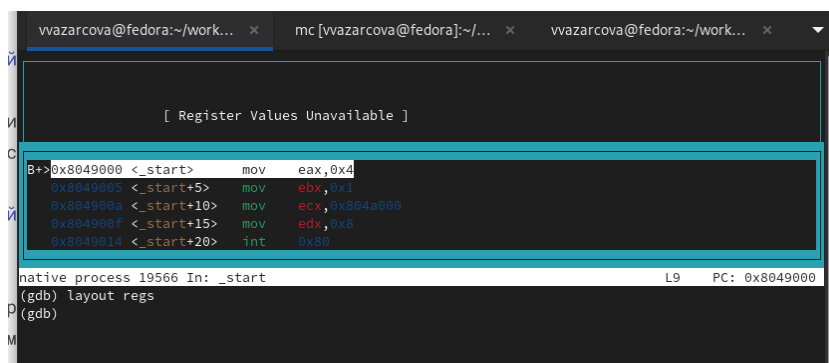


Рис. 3.10: Режим псевдографики

На предыдущих шагах была установлена точка останова по имени метки (_start). Проверяю это с помощью команды info breakpoints (кратко i b) (рис. 3.11).

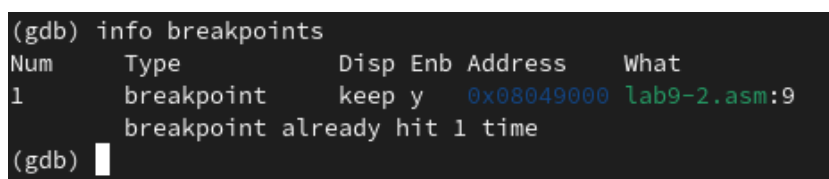


Рис. 3.11: Проверка меток

Устанавливаю ещё одну точку останова по адресу инструкции и проверяю (рис. 3.12).

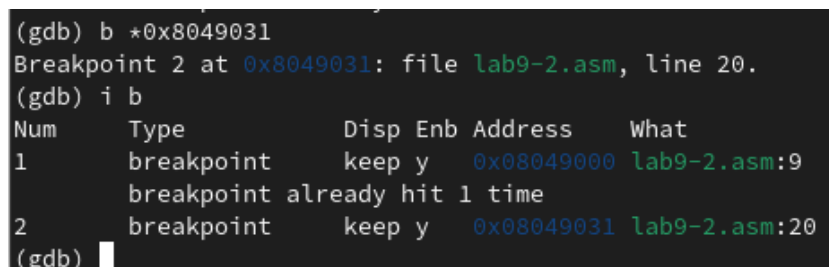


Рис. 3.12: Установка второй метки и проверка

Выполняю 5 инструкций с помощью команды stepi (или si) и слежу за изменением значений регистров. Значения регистров ebx, ecx, edx и eax изменялись (рис. 3.13).

The screenshot shows a GDB window with the following content:

Register group: general		
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xffffd0a0	0xffffd0a0
ebp	0x0	0x0
esi	0x0	0


```

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
  
```

native process 19566 In: _start L14 PC: 0x8049016

```

1 breakpoint keep y 0x8049000 lab9-2.asm:9
  breakpoint already hit 1 time
2 breakpoint keep y 0x8049031 lab9-2.asm:20
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)
  
```

Рис. 3.13: 5 инструкций с помощью stepi

Смотрю значение переменной msg1 по имени (рис. 3.14).

```

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

Рис. 3.14: Просмотр значения переменной

Изменяю первый символ переменной msg1 и msg2 (рис. 3.15).

```

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

Рис. 3.15: Изменение символов в переменных

Выведу в разных форматах значение регистра edx (рис. 3.16).

```

(gdb) p/s $edx
$3 = 8
(gdb) p/t $edx
$4 = 1000
(gdb) p/x $edx
$5 = 0x8
(gdb)

```

Рис. 3.16: Значение регистра edx в разных форматах

С помощью команды set измените значение регистра ebx (рис. 3.17).

```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb)

```

Рис. 3.17: Смена значения регистра ebx

Завершаю выполнение программы.

3. Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки в файл именем lab9-3.asm, создаю исполняемый файл, и загружаю его в отладчик с ключем `--args` (рис. 3.18).

```

vvazarcova@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab9-3.asm
vvazarcova@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
vvazarcova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
vvazarcova@fedora:~/work/arch-pc/lab09$ gbd --args lab9-3 аргумент1 аргумент 2 'аргумент 3

```

Рис. 3.18: Копирование, создание исполняемого файла, загрузка

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее (рис. 3.19).

```

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

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

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

```

Рис. 3.19: Запуск с точкой останова

Посмотрю 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 (рис. 3.20).

```

(gdb) x/x $esp
0xffffd070: 0x00000005
(gdb) x/s *(void*)($esp+4)
0xffffd231: "/home/vvazarcova/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void*)($esp+8)
0xffffd25c: "аргумент1"
(gdb) x/s *(void*)($esp+12)
0xffffd26e: "аргумент"
(gdb) x/s *(void*)($esp+16)
0xffffd27f: "2"
(gdb) x/s *(void*)($esp+20)
0xffffd281: "аргумент 3"
(gdb) x/s *(void*)$esp+24
Junk after end of expression.
(gdb) x/s *(void*)($esp+24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 3.20: Просмотр позиций стека

Шаг 4 обусловлен разрядностью системы, а указатель `void` занимает 4 байта.

4 Задания для самостоятельной работы

1. Преобразую программу из лабораторной работы 8, реализоваа вычисление функции как подпрограмму.

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

```
SECTION .data
```

```
msg_func db "Функция:  $f(x) = 10x - 4$ ", 0
```

```
msg_result db "Результат: ", 0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov eax, msg_func
```

```
call sprintf
```

```
pop ecx
```

```
pop edx
```

```
sub ecx, 1
```

```
mov esi, 0
```

```

next:
cmp ecx, 0h
jz _end
pop eax
call atoi

call _calculate_fx

add esi, eax
loop next

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

_calculate_fx:
mov ebx, 10
mul ebx
sub eax, 4

```

2. Запускаю программу в режиме отладчика и просмотрю изменение значений регистров. При выполнении инструкции `mul ecx` результат записывается в `eax`, но также меняет `edx`. Значение регистра `ebx` не обновляется, поэтому результат программа неверно подсчитывает функцию. Исправляю это и функция работает корректно.

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

```
SECTION .data
```

```
div: DB 'Результат: ', 0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov ebx, 3
```

```
mov eax, 2
```

```
add ebx, eax
```

```
mov eax, ebx
```

```
mov ecx, 4
```

```
mul ecx
```

```
add eax, 5
```

```
mov edi, eax
```

```
mov eax, div
```

```
call sprint
```

```
mov eax, edi
```

```
call iprintLF
```

```
call quit
```


5 Выводы

Подводя итоги данной лабораторной работы, я получила навыки написания программ с использованием подпрограмм, а так же познакомилась с методами отладки при помощи GDB.

Список литературы