

Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютеров

Лазарев Даниил Михайлович

Содержание

1	Цель работы	4
2	Теоретическое введение	5
3	Выполнение лабораторной работы	6
4	Выполнение самостоятельной работы	18
5	Выводы	21

Список иллюстраций

3.1	Создание файла в каталоге	6
3.2	Код программы в файле	7
3.3	Преобразование в исполняемый файл	7
3.4	Измененный код программы	8
3.5	Преобразование измененного файла	9
3.6	Текст в файле	9
3.7	Преобразование файла в исполняемый	10
3.8	Файл в gdb	10
3.9	Установка брейкпоинта	10
3.10	Просмотр кода с помощью disassemble	11
3.11	Отображение команд	12
3.12	Проверка брейкпоинтов	14
3.13	Значение переменной	14
3.14	Значение переменной	14
3.15	Изменение значения переменной	15
3.16	Изменение значения переменной	15
3.17	Изменение значения регистра	15
3.18	Копирование файла	16
3.19	Преобразование файла в исполняемый	16
3.20	Файл в gdb	16
3.21	Количество аргументов	17
4.1	Преобразование программы	18
4.2	Код в файле	19
4.3	Исправленный файл	20

1 Цель работы

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

- обнаружение ошибки; • поиск её местонахождения; • определение причины ошибки; • исправление ошибки. Можно выделить следующие типы ошибок:
- синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • семантические ошибки — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата; • ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

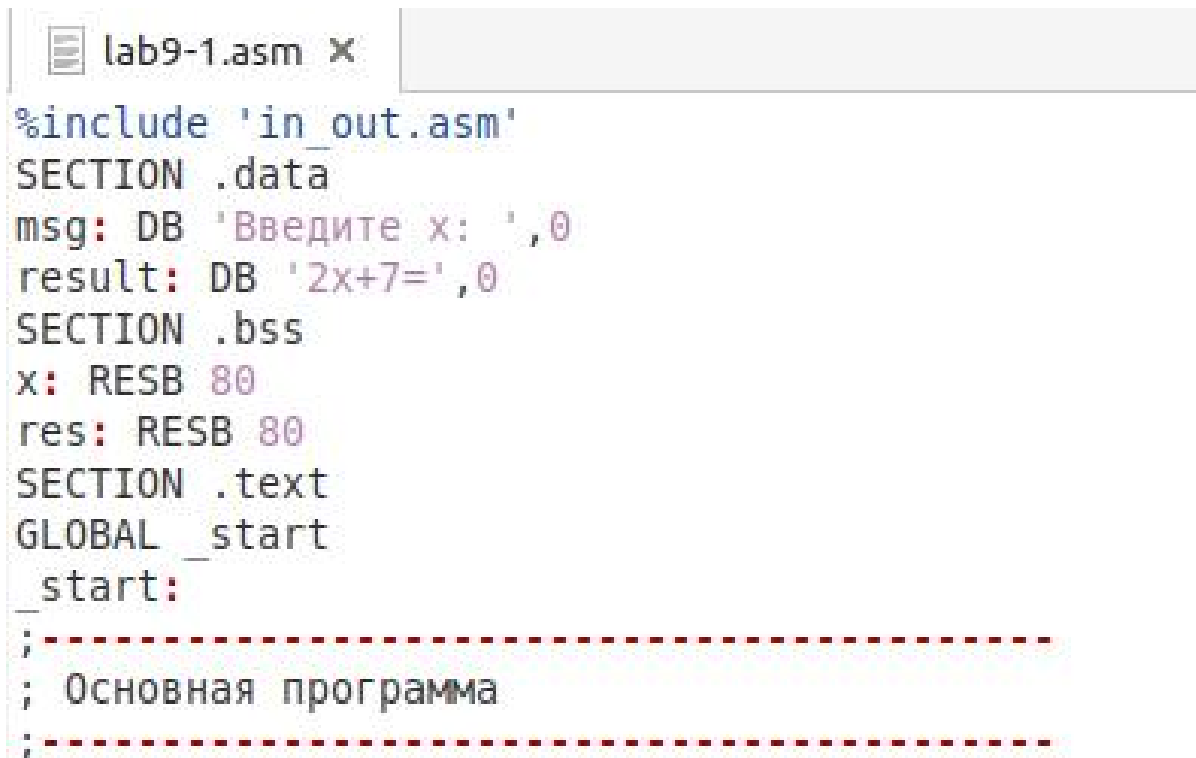
3 Выполнение лабораторной работы

Создадим каталог для программ лаб. работы н.9, перейдем в него и создадим файл "lab9-1.asm" (рис. @fig:1)

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

Рис. 3.1: Создание файла в каталоге

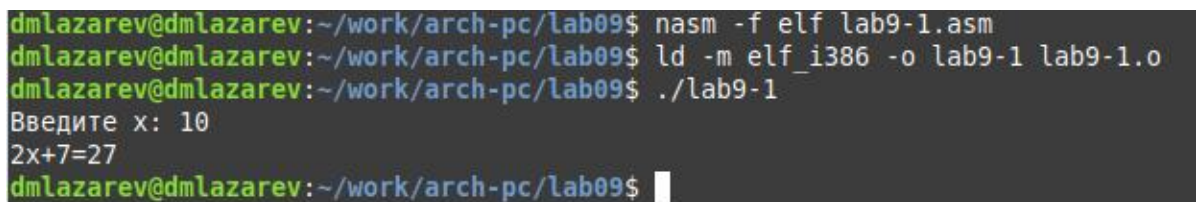
Введем в созданный файл текст программы из предложенного нам листинга 9.1 (рис. @fig:2)



```
lab9-1.asm x
%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
;-----
```

Рис. 3.2: Код программы в файле

Создадим исполняемый файл и запустим его, предварительно скопировав из предыдущей



```
dmlazarev@dmlazarev:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
dmlazarev@dmlazarev:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
dmlazarev@dmlazarev:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 10
2x+7=27
dmlazarev@dmlazarev:~/work/arch-pc/lab09$
```

Рис. 3.3: Преобразование в исполняемый файл

Далее дополним код так, чтобы X проходил еще через одну функцию. (рис. @fig:4)

```
; -----  
; Подпрограмма вычисления  
; выражения "2x+7"  
_calcul:  
call _subcalcul  
mov ebx,2  
mul ebx  
add eax,7  
mov [res],eax  
ret ; выход из подпрограммы  
; -----  
_subcalcul:  
mov ebx,3  
mul ebx  
add eax,-1  
mov [res],eax  
ret
```

Рис. 3.4: Измененный код программы

Преобразуем в исполняемый файл и проверим правильность выполнения. (рис. @fig:5)


```

dmlazarev@dmlazarev:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
dmlazarev@dmlazarev:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
dmlazarev@dmlazarev:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 5
2x+7=35
dmlazarev@dmlazarev:~/work/arch-pc/lab09$

```

Рис. 3.5: Преобразование измененного файла

Создадим файл "lab9-2.asm" и вставим в него предложенный нам листинг 9.2 (рис. 9.2).

```

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

```

Рис. 3.6: Текст в файле

Преобразуем файл "lab9-2.asm" в исполняемый, добавив в него отладочную информацию.

```

dmlazarev@dmlazarev:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
dmlazarev@dmlazarev:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
dmlazarev@dmlazarev:~/work/arch-pc/lab09$ gdb lab9-2
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.

```

Рис. 3.7: Преобразование файла в исполняемый

Загрузим исполняемый файл в отладчик gdb и запустим программу с помощью команды "run".

```

(gdb) run
Starting program: /home/dmlazarev/work/arch-pc/lab09/lab9-2
Hello, world!
Inferior 1 (process 7165) exited normally
(gdb)

```

Рис. 3.8: Файл в gdb

Установим брейкпоинт с помощью команды "break" для более подробного анализа и запуска программы.

```

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

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

```

Рис. 3.9: Установка брейкпоинта

Посмотрим код с помощью команды "disassemble".(рис. @fig:10)

```

(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     $0x4, %eax
    0x08049005 <+5>:      mov     $0x1, %ebx
    0x0804900a <+10>:     mov     $0x04a000, %ecx
    0x0804900f <+15>:     mov     $0x8, %edx
    0x08049014 <+20>:     int     $0x80
    0x08049016 <+22>:     mov     $0x4, %eax
    0x0804901b <+27>:     mov     $0x1, %ebx
    0x08049020 <+32>:     mov     $0x04a000, %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. 3.10: Просмотр кода с помощью disassemble

Включим отображение команд на синтаксисе Интела.(рис. @fig:11)

```

(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. 3.11: Отображение команд

Включим режим псевдографики для более удобного анализа программы.(рис. @fig:12; p

```
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 7331 In:  start                                L9    PC: 0x8049000
(gdb) 
```

```
[ 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

native process 7331 In:  start                                L9    PC: 0x8049000
(gdb) layout regs
(gdb) 
```

Проверим все установленные брейкпоинты и установим еще одну точку остановки по ад

```
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

native process 7331 In:  start                                L9    PC: 0x8049000
Note: breakpoint 1 also set at pc 0x8049000.
Breakpoint 2 at 0x8049000: file lab9-2.asm, line 9.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab9-2.asm:9
         breakpoint already hit 1 time
2        breakpoint      keep y   0x08049000 lab9-2.asm:9
(gdb)
```

Рис. 3.12: Проверка брейкпоинтов

Посмотрим значение переменной msg1 по имени. (рис. @fig:15)

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

Рис. 3.13: Значение переменной

Посмотрим значение переменной msg2 по адресу. (рис. @fig:16)

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

Рис. 3.14: Значение переменной

Изменим первый символ переменной msg1. (рис. @fig:17)

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

Рис. 3.15: Изменение значения переменной

Так же изменим второй символ во второй переменной msg2 (рис. 3.16)

```
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb &msg2
0x804a008 <msg2>: "Lor d!\n\034"
(gdb)
```

Рис. 3.16: Изменение значения переменной

С помощью команды "set" изменим значение регистра ebx. Разница вывода в том, что

```
0x804a008 <msg2>: "Lor d!\n\034"
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb)
```

Рис. 3.17: Изменение значения регистра

Скопируем файл "lab8-2.asm" созданный при выполнении предыдущей лабораторной работы в файл "lab09-3.asm" (рис. @fig:20)

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

Рис. 3.18: Копирование файла

Преобразуем файл в исполняемый. (рис. @fig:21)

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

Рис. 3.19: Преобразование файла в исполняемый

Загрузим программу в gdb с использованием ключа --args и указав 3 аргумента. Установим точку останова в начале программы.

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

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

Рис. 3.20: Файл в gdb

Введя команду "x/x \$esp" увидим то, что аргументов командной строки 4(включая название файла).


```
(gdb) x/x $esp  
0xffffd200: 0x00000004  
(gdb) █
```

Рис. 3.21: Количество аргументов

4 Выполнение самостоятельной работы

Преобразуем программу из предыдущей лабораторной работы реализовав вычисление зна

Преобразование программы

Рис. 4.1: Преобразование программы

Создадим файл “lab9-4.asm” и внесем в него листинг 9.3. С помощью gdb проанализируем и найдем ошибку в коде. (рис. 4.2)

```

#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. 4.2: Код в файле

Исправим ошибку в коде. (рис. 4.3)

```

#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
; ----
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 4.3: Исправленный файл

5 Выводы

В ходе лабораторной работы мы научились написанию программ с использованием подпрограмм, а так же с отладкой программ с помощью GDB.