

Отчет по лабораторной работе №9

Дисциплина: Архитектура компьютера

Краснова Камилла Геннадьевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Реализация подпрограмм в NASM	9
4.2	Отладка программ с помощью GDB	10
4.3	Добавление точек останова	14
4.4	Работа с данными программы в GDB	15
4.5	Обработка аргументов командной строки в GDB	17
4.6	Задание для самостоятельной работы	18
5	Выводы	23
6	Список литературы	24

Список иллюстраций

4.1	Создание директории	9
4.2	Редактирование файла	9
4.3	Запуск исполняемого файла	10
4.4	Редактирование файла	10
4.5	Запуск исполняемого файла	10
4.6	Создание файла	11
4.7	Редактирование файла	11
4.8	Получение исполняемого файла	11
4.9	Загрузка файла	11
4.10	Проверка работы программы	12
4.11	Запуск отладчика с брекпоинтом	12
4.12	Дисассимилирование программы	12
4.13	Переключение отображения команд	13
4.14	Режим псевдографики	13
4.15	Режим псевдографики	14
4.16	Проверка установки точки	14
4.17	Добавление точки останова	15
4.18	Просмотр содержимого	15
4.19	Просмотр переменных	16
4.20	Просмотр значения регистра разными представлениями	16
4.21	Примеры использования команды set	17
4.22	Подготовка новой программы	17
4.23	Запуск в режиме отладки	17
4.24	Запуск отладки	18
4.25	Проверка работы стека	18
4.26	Измененная программа предыдущей лабораторной работы	19
4.27	Поиск ошибки в программе через пошаговую отладку	21
4.28	Проверка корректировок в программе	21

Список таблиц

1 Цель работы

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

2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Самостоятельное выполнение заданий по материалам лабораторной работы

3 Теоретическое введение

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

- 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, отрабатывает, 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).

Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

Наиболее часто применяют следующие методы отладки: • создание точек кон-

троля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);

- использование специальных программ-отладчиков. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам.

Пошаговое выполнение — это выполнение программы с остановкой после каждой строки, чтобы программист мог проверить значения переменных и выполнить другие действия. Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова: • Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом);

- Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его).

Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

С помощью утилиты `mkdir` создаю директорию, в которой буду создавать файлы с программами для лабораторной работы №9. Перехожу в созданный каталог с помощью утилиты `cd` и с помощью утилиты `touch` создаю файл `lab09-1.asm`. (рис. 4.1).

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

Рис. 4.1: Создание директории

Открываю созданный файл `lab09-1.asm`, вставляю в него программу (рис. 4.2).

```
kamilla@fedora:~/work/arch-pc/lab09$ gedit lab09-1.asm
Открыть ▼ + *lab09-1.asm
~/work/arch-pc/lab09
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg: DB 'Введите x: ',0
5 result: DB '2x+7=',0
6 SECTION .bss
7 x: RESB 80
8 res: RESB 80
9 SECTION .text
10 GLOBAL _start
11 _start:
12
```

Рис. 4.2: Редактирование файла

Создаю исполняемый файл программы и запускаю его (рис. 4.3).

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

Рис. 4.3: Запуск исполняемого файла

Изменяю текст программы, добавляя подпрограмму `_subcalcul` в подпрограмму `_calcul`(рис. 4.4).

```
15 mov eax, msg
16 call sprint
17 mov ecx, x
18 mov edx, 80
19 call sread
20 mov eax, x
21 call atoi
22 call _calcul ; Вызов подпрограммы _calcul
23 mov eax, result
24 call sprint
25 mov eax, [res]
26 call iprintLF
27 call quit
28 ;-----
29 ; Подпрограмма вычисления
30 ; выражения "2x+7"
31 _calcul:
32 push eax
33 call _subcalcul
34 mov ebx, 7
```

Рис. 4.4: Редактирование файла

Создаю новый исполняемый файл программы и запускаю его (рис. 4.5).

```
kamilla@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
kamilla@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
kamilla@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 2
2(3x-1)+7=17
kamilla@fedora:~/work/arch-pc/lab09$
```

Рис. 4.5: Запуск исполняемого файла

4.2 Отладка программ с помощью GDB

Создаю файл `lab09-2.asm` (рис. 4.6).

Проверяю работу программы, запустив ее в оболочке GDB (рис. 4.10).

```
(gdb) run
Starting program: /home/kamilla/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 9530) exited normally]
(gdb)
```

Рис. 4.10: Проверка работы программы

Устанавливаю брекпоинт на метку `_start` и запускаю программу (рис. 4.11).

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

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb)
```

Рис. 4.11: Запуск отладчика с брекпоинтом

Далее смотрю дисассимилированный код программы (рис. 4.12).

```
(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. 4.12: Дисассимилирование программы

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 4.13).

```

(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. 4.13: Переключение отображения команд

Включаю режим псевдографики (рис. 4.14) и (рис. 4.15).

```

B> 0x08049000 <_start>  mov     eax,0x4
    0x08049005 <_start+5>  mov     ebx,0x1
    0x0804900a <_start+10> mov     ecx,0x804a000
    0x0804900f <_start+15> mov     edx,0x8
    0x08049014 <_start+20> int     0x80
    0x08049016 <_start+22> mov     eax,0x4
    0x0804901b <_start+27> mov     ebx,0x1
    0x08049020 <_start+32> mov     ecx,0x804a008
    0x08049025 <_start+37> mov     edx,0x7
    0x0804902a <_start+42> int     0x80
    0x0804902c <_start+44> mov     eax,0x1
    0x08049031 <_start+49> mov     ebx,0x0
    0x08049036 <_start+54> int     0x80
native process 9572 In: _start          L9    PC: 0x08049000
(gdb)

```

Рис. 4.14: Режим псевдографики

```

[ 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

native process 9572 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 4.15: Режим псевдографики

Различия между синтаксисом АТТ и Intel заключаются в порядке операндов (АТТ - Операнд источника указан первым. Intel - Операнд назначения указан первым), их размере (АТТ - размер операндов указывается явно с помощью суффиксов, непосредственные операнды предваряются символом \$; Intel - Размер операндов неявно определяется контекстом, как ах, еах, непосредственные операнды пишутся напрямую), именах регистров (АТТ - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов).

4.3 Добавление точек останова

Проверяю установку точки останова по имени метки (рис. 4.16).

```

native process 9572 In: _start L9
(gdb) layout regs
(gdb) info breakpoints
Num      Type          Disp Enb Address      What
1        breakpoint    keep y   0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb)

```

Рис. 4.16: Проверка установки точки

Устанавливаю точку останова по адресу инструкции (рис. 4.17).

```

(gdb) break *0x08049031
Breakpoint 2 at 0x08049031: file lab09-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
2        breakpoint      keep y   0x08049031 lab09-2.asm:20
(gdb) 

```

Рис. 4.17: Добавление точки останова

4.4 Работа с данными программы в GDB

Просматриваю содержимое регистров командой `info registers` (рис. 4.18).

```

0x80496f6      add     BYTE PTR [eax],al
0x80496f8      add     BYTE PTR [eax],al
0x80496fa      add     BYTE PTR [eax],al
0x80496fc      add     BYTE PTR [eax],al
0x80496fe      add     BYTE PTR [eax],al
0x8049700      add     BYTE PTR [eax],al
0x8049702      add     BYTE PTR [eax],al
0x8049704      add     BYTE PTR [eax],al
0x8049706      add     BYTE PTR [eax],al
0x8049708      add     BYTE PTR [eax],al

native process 9572 In: _start
eax            0x0            0
ecx            0x0            0
edx            0x0            0
ebx            0x0            0
esp            0xffffd0d0     0xffffd0d0
ebp            0x0            0x0
esi            0x0            0
edi            0x0            0
eip            0x8049000     0x8049000 <_start>
eflags         0x202         [ IF ]
cs             0x23          35
ss             0x2b          43
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.18: Просмотр содержимого

Смотрю содержимое переменных по имени и по адресу (рис. 4.19).

```

0x80496f6      add     BYTE PTR [eax],al
0x80496f8      add     BYTE PTR [eax],al
0x80496fa      add     BYTE PTR [eax],al
0x80496fc      add     BYTE PTR [eax],al
0x80496fe      add     BYTE PTR [eax],al
0x8049700      add     BYTE PTR [eax],al
0x8049702      add     BYTE PTR [eax],al
0x8049704      add     BYTE PTR [eax],al
0x8049706      add     BYTE PTR [eax],al
0x8049708      add     BYTE PTR [eax],al

native process 9572 In: _start
esi           0x0           0
edi           0x0           0
eip           0x8049000      0x8049000 <_start>
eflags       0x202         [ IF ]
cs            0x23          35
ss            0x2b          43
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)

```

Рис. 4.19: Просмотр переменных

Меняю содержимое переменных по имени и по адресу (рис. ??) и (рис. ??).

<pre> (gdb) set {char}&msg1='h' (gdb) x/1sb &msg1 0x804a000 <msg1>: "hello, " (gdb) </pre>	<pre> (gdb) set {char}&msg2='r' (gdb) x/1sb &msg2 0x804a008 <msg2>: "rorl" (gdb) </pre>
---	--

Вывожу в различных форматах значение регистра edx (рис. 4.20).

```

(gdb) p/s $edx
$1 = 0
(gdb) p/t $edx
$2 = 0
(gdb) p/x $edx
$3 = 0x0

```

Рис. 4.20: Просмотр значения регистра разными представлениями

С помощью команды set меняю содержимое регистра ebx (рис. 4.21).


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

Рис. 4.21: Примеры использования команды set

4.5 Обработка аргументов командной строки в GDB

Копирую программу из предыдущей лабораторной работы в текущий каталог и создаю исполняемый файл с файлом листинга и отладки (рис. 4.22).

```
kamilla@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
kamilla@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
kamilla@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Рис. 4.22: Подготовка новой программы

Запускаю программу с режиме отладки с указанием аргументов (рис. 4.23).

```
kamilla@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 1 2 3
GNU gdb (Fedora Linux) 14.2-1.fc40
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)
```

Рис. 4.23: Запуск в режиме отладки

Указываю брейкпоинт и запускаю отладку. (рис. ??).

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 7.
(gdb) run
Starting program: /home/kamilla/work/arch-pc/lab09/lab09-3 1 2 3

This GDB supports auto-downloading debuginfo from the following URLs:
  <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 .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000

Breakpoint 1, _start () at lab09-3.asm:7
7      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)

```

Рис. 4.24: Запуск отладки

Проверяю работу стека, изменяя аргумент команды просмотра регистра esp на +4, число обусловлено разрядностью системы, а указатель void занимает как раз 4 байта. Ошибка при аргументе +20 означает, что аргументы на вход программы закончились. (рис. 4.25).

```

(gdb) x/s *(void**)(esp + 4)
0xffffd279:  "/home/kamilla/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd2a2:  "1"
(gdb) x/s *(void**)(esp + 12)
0xffffd2a4:  "2"
(gdb) x/s *(void**)(esp + 16)
0xffffd2a6:  "3"
(gdb) x/s *(void**)(esp + 20)
0x0:  <error: Cannot access memory at address 0x0>
(gdb) x/s *(void**)(esp + 24)
0xffffd2a8:  "SHELL=/bin/bash"
(gdb)

```

Рис. 4.25: Проверка работы стека

4.6 Задание для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. 4.26).

```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg_func db "Функция: f(x) = 10(x-1)", 0
5 msg_result db "Результат: ", 0
6
7 SECTION .text
8 GLOBAL _start
9
10 _start:
11 mov eax, msg_func
12 call sprintfLF
13
14 pop ecx
15 pop edx
```

Рис. 4.26: Измененная программа предыдущей лабораторной работы

Код программы:

```
%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 sprintfLF

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 просматриваю изменение значений регистров через `i r`. При выполнении инструкции `mul ecx` можно заметить, что результат умножения записывается в регистр `eax`, но также меняет и `edx`. Значение регистра `ebx` не обновляется напрямую, поэтому программа неверно подсчитывает функцию (рис. 4.27).

```

B> 0x80490e8 <_start> mov ebx,0x3
    0x80490ed <_start+5> mov eax,0x2
    0x80490f2 <_start+10> add ebx,eax
    0x80490f4 <_start+12> mov ecx,0x4
    0x80490f9 <_start+17> mul ecx
    0x80490fb <_start+19> add ebx,0x5
    0x80490fe <_start+22> mov edi,ebx
    0x8049100 <_start+24> mov eax,0x804a000
    0x8049105 <_start+29> call 0x804900f <sprint>
    0x804910a <_start+34> mov eax,edi
    0x804910c <_start+36> call 0x8049086 <iprintf>
    0x8049111 <_start+41> call 0x80490db <quit>
    0x8049116 add BYTE PTR [eax],al
    0x8049118 add BYTE PTR [eax],al
    0x804911a add BYTE PTR [eax],al
    0x804911c add BYTE PTR [eax],al
    0x804911e add BYTE PTR [eax],al
    0x8049120 add BYTE PTR [eax],al
    0x8049122 add BYTE PTR [eax],al
    0x8049124 add BYTE PTR [eax],al
    0x8049126 add BYTE PTR [eax],al
    0x8049128 add BYTE PTR [eax],al

native process 21975 In: _start
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd0c0 0xffffd0c0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490e8 0x80490e8 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.27: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. 4.28).

```

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

```

Рис. 4.28: Проверка корректировок в программе

Код измененной программы:

```

#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

При выполнении данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм, а так же познакомилась с методами отладки при помощи GDB и его основными возможностями.

6 Список литературы

1. Лабораторная работа №9