

Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютера

Репкина Елизавета Андреевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	26
	Список литературы	27

Список иллюстраций

4.1	Выполнение команд	9
4.2	Редактирование файла	9
4.3	Запуск файла	10
4.4	Редактирование файла	10
4.5	Запуск файла	10
4.6	Создание файла печати сообщения Hello world!	11
4.7	Получение и загрузка в отладчик исполняемого файла	12
4.8	Просмотр дисассимилированного кода программы	12
4.9	Просмотр дисассимилированного кода программы	13
4.10	Переключение на Intel'ский синтаксис	13
4.11	Включение режима псевдографики	14
4.12	Проверка точки останова	14
4.13	Установка точки останова и её проверка	15
4.14	До использования команды stepi	16
4.15	После использования команды stepi	17
4.16	Просмотр значения переменной msg1	17
4.17	Просмотр значения переменной msg2	17
4.18	Изменение первого символа msg1	18
4.19	Изменение первого символа переменной msg2	18
4.20	Вывод значений регистра ebx	18
4.21	Изменение значения регистра ebx	19
4.22	Копирование файла	19
4.23	Создание исполняемого файла	19
4.24	Загрузка исполняемого файла в отладчик	20
4.25	Установка точки останова	20
4.26	Просмотр вершины стека	20
4.27	Просмотр остальных позиций стека	21
4.28	Редактирование файла	21
4.29	Запуск исполняемого файла	22
4.30	Редактирование файла	23
4.31	Запуск исполняемого файла	23
4.32	Просмотр программы	24
4.33	Редактирование программы	24
4.34	Запуск исполняемого файла	25

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. Релизация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Добавление точек останова
4. Работа с данными программы в GDB
5. Обработка аргументов командной строки в GDB
6. Задание для самостоятельной работы.

3 Теоретическое введение

Понятие об отладке

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, 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).

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.

Методы отладки

Наиболее часто применяют следующие методы отладки:

- создание точек контроля 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);

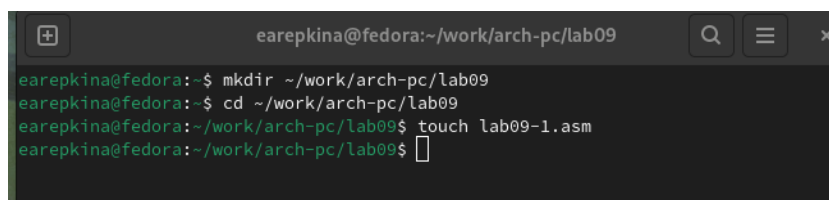
- Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его).

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

Реализация подпрограмм в NASM

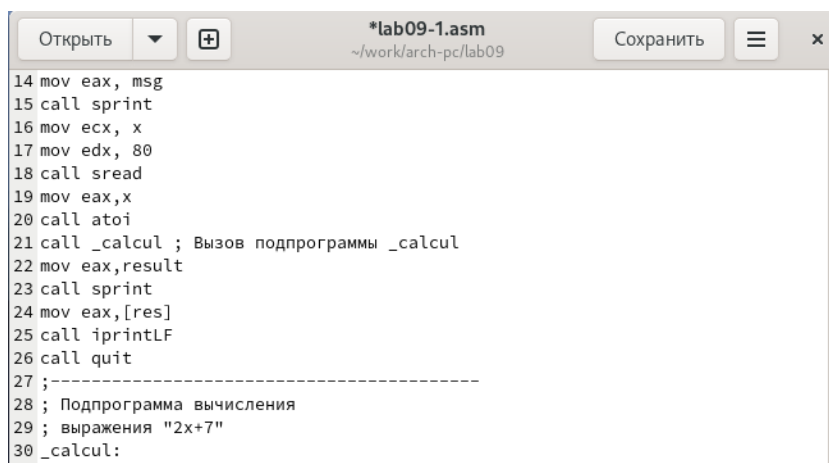
Создаю каталог для выполнения лабораторной работы №9, перехожу в него и создаю файл lab09-1.asm.(рис. 4.1)



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

Рис. 4.1: Выполнение команд

Ввожу в файл lab09-1.asm текст программы из листинга 9.1. (рис. 4.2)



```
*lab09-1.asm
~/work/arch-pc/lab09

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

Рис. 4.2: Редактирование файла

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

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

Рис. 4.3: Запуск файла

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



```
*lab09-1.asm
~/work/arch-pc/lab09
Сохранить

20 call atoi
21 call _calcul ; Вызов подпрограммы _calcul
22 mov eax,result
23 call sprint
24 mov eax,[res]
25 call iprintLF
26 call quit
27 ;-----
28 ; Подпрограмма вычисления
29 ; выражения "2x+7"
30 _calcul:
31 call _subcalcul
32 mov ebx,2
33 mul ebx
34 add eax,7
35 mov [res],eax
36 ret
37 _subcalcul:
38 mov ebx,3
39 mul ebx
40 sub eax,1
41 ret
```

Рис. 4.4: Редактирование файла

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

```
earepkina@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
earepkina@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
earepkina@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 10
2x+7=65
earepkina@fedora:~/work/arch-pc/lab09$
```

Рис. 4.5: Запуск файла

Программа работает корректно.

Отладка программ с помощью GDB

Создаю файл `lab09-2.asm` с текстом программы из Листинга 9.2 (рис. 4.6)



```
1 SECTION .data
2 msg1: db "Hello, ",0x0
3 msg1Len: equ $ - msg1
4 msg2: db "world!",0xa
5 msg2Len: equ $ - msg2
6 SECTION .text
7 global _start
8 _start:
9 mov eax, 4
10 mov ebx, 1
11 mov ecx, msg1
12 mov edx, msg1Len
13 int 0x80
14 mov eax, 4
15 mov ebx, 1
16 mov ecx, msg2
17 mov edx, msg2Len
18 int 0x80
19 mov eax, 1
20 mov ebx, 0
21 int 0x80
```

Рис. 4.6: Создание файла печати сообщения Hello world!

Получаю исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'. Далее загружаю исполняемый файл в отладчик gdb и запускаю его с помощью команды run. (рис. 4.7)

```
earepkina@fedora:~/work/arch-pc/lab09
earepkina@fedora:~/work/arch-pc/lab09$ cd ~/work/arch-pc/lab09
earepkina@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
earepkina@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
earepkina@fedora:~/work/arch-pc/lab09$ gdb lab09-2
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 lab09-2...
(gdb) r
Starting program: /home/earepkina/work/arch-pc/lab09/lab09-2

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.
Hello, world!
[Inferior 1 (process 7408) exited normally]
```

Рис. 4.7: Получение и загрузка в отладчик исполняемого файла

Устанавливаю брейкпоинт на метку `_start` и запускаю программу (рис. 4.8)

```
[Inferior 1 (process 7408) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) r
Starting program: /home/earepkina/work/arch-pc/lab09/lab09-2

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

Рис. 4.8: Просмотр дисассимилированного кода программы

Смотрю дисассимилированный код программы с помощью команды `disassemble` начинаю с метки `_start` (рис. 4.9)

```
(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. 4.9: Просмотр дисассимилированного кода программы

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 4.10)

```
(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. 4.10: Переключение на Intel'ский синтаксис

Отличия заключаются в том, что в режиме АТТ используются “%” перед именами регистров и “\$” перед именами операндов, а в режиме Intel используется обычный синтаксис. Включаю режим псевдографики для более удобного анализа программы(рис. 4.11)

```

[ Register Values Unavailable ]

0x80490b4  add  BYTE PTR [eax],al
0x80490b6  add  BYTE PTR [eax],al
0x80490b8  add  BYTE PTR [eax],al
0x80490ba  add  BYTE PTR [eax],al
0x80490bc  add  BYTE PTR [eax],al
0x80490be  add  BYTE PTR [eax],al
0x80490c0  add  BYTE PTR [eax],al

native process 4304 In: _start          L9  PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 4.11: Включение режима псевдографики

Добавление точек останова Проверяю точку останова по имени метки (рис. 4.12)

```

(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab09-2.asm:9

```

Рис. 4.12: Проверка точки останова

Определяю адрес предпоследней инструкции и устанавливаю точку останова. Далее смотрю информацию о всех установленных точках останова (рис. 4.13)

```
0x8049016 <_start+22>  mov    $0x4,%eax
0x804901b <_start+27>  mov    $0x1,%ebx
0x8049020 <_start+32>  mov    $0x804a008,%ecx
0x8049025 <_start+37>  mov    $0x7,%edx
0x804902a <_start+42>  int    $0x80
0x804902c <_start+44>  mov    $0x1,%eax
b+ 0x8049031 <_start+49>  mov    $0x0,%ebx
0x8049036 <_start+54>  int    $0x80

exec No process in:
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y  0x08049000  lab09-2.asm:9
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y  0x08049000  lab09-2.asm:9
2        breakpoint      keep y  0x08049031  lab09-2.asm:20
(gdb) █
```

Рис. 4.13: Установка точки останова и её проверка

Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды `stepi` и слежу за изменением регистров. (рис. 4.14) (рис. 4.15)

```

eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffc320 0xffffc320  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049000 0x8049000 <_start>  eflags   0x202    [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43
fs       0x0      0      gs       0x0      0

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
b+ 0x8049036 <_start+54> int     0x80
0x8049038 add     BYTE PTR [eax],al
0x804903a add     BYTE PTR [eax],al

native process 4883 In: _start L9 P
(gdb) i r
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc320 0xffffc320
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb)

```

Рис. 4.14: До использования коадны stepi


```

eax      0x8      8      ecx      0x804a000      134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffc320      0xffffc320      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016      0x8049016 <_start+22>      eflags      0x202      [ IF ]
cs       0x23      35      ss       0x2b      43
ds       0x2b      43      es       0x2b      43
fs       0x0      0      gs       0x0      0

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
b+ 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 4883 In: _start      L14
(gdb) i r
eax      0x8      8
ecx      0x804a000      134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc320      0xffffc320
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016      0x8049016 <_start+22>
eflags   0x202      [ IF ]
cs       0x23      35
ss       0x2b      43
ds       0x2b      43
es       0x2b      43
fs       0x0      0
gs       0x0      0

```

Рис. 4.15: После использования команды stepi

Изменились регистры eax,ebx,ecx,edx. Просматриваю значение переменной msg1 по имени (рис. 4.16)

```

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

```

Рис. 4.16: Просмотр значения переменной msg1

Также просматриваю значение переменной msg2 (рис. 4.17)

```

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

```

Рис. 4.17: Просмотр значения переменной msg2

Изменяю первый символ переменной msg1 (рис. 4.18)

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

Рис. 4.18: Изменение первого символа msg1

Изменяю первый символ переменной msg2. (рис. 4.19)

```
(gdb) set {char}0x804a008='t'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "torld!\n\034"
(gdb)
```

Рис. 4.19: Изменение первого символа переменной msg2

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

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

Рис. 4.20: Вывод значений регистра ebx

С помощью команды set изменяю значение регистра ebx(рис. 4.21)

```

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

```

Рис. 4.21: Изменение значения регистра ebx

Обработка аргументов командной строки в GDB

Копирую 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. 4.22)

```

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

```

Рис. 4.22: Копирование файла

Создаю исполняемый файл (рис. 4.23)

```

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

```

Рис. 4.23: Создание исполняемого файла

Загружаю исполняемый файл в отладчик, указав аргументы (рис. 4.24)

```
earepkina@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 '
аргумент 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. 4.24: Загрузка исполняемого файла в отладчик

Для начала устанавливаю точку останова перед первой инструкцией в программе и запускаю её (рис. 4.25)

```
Reading symbols from lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/earepkina/work/arch-pc/lab09/lab09-3 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 lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 4.25: Установка точки останова

Просматриваю вершину стека, то есть число аргументов строки(включая имя программы) (рис. 4.26)

```
(gdb) x/x $esp
0xffffd070: 0x00000005
(gdb)
```

Рис. 4.26: Просмотр вершины стека

Просматриваю остальные позиции стека. (рис. 4.27)

```

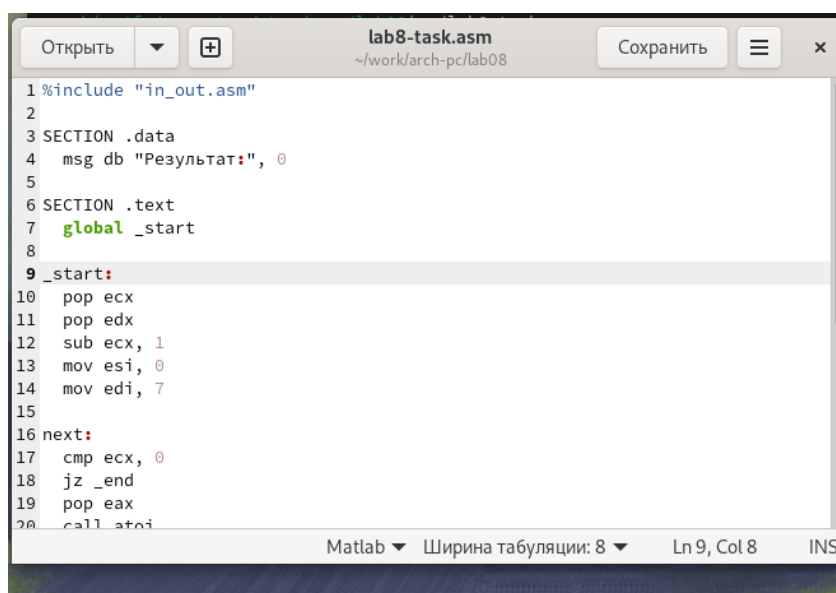
(gdb) x/s *(void**)($esp + 4)
0xffffd230: "/home/earepkina/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd25b: "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd26d: "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd27e: "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd280: "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 4.27: Просмотр остальных позиций стека

Шаг изменения адреса равен 4, потому что значение регистра esp в стеке увеличивается на 4

Задание для самостоятельной работы. 1. Открываю программу из лабораторной работы №8 и начинаю её редактировать (рис. 4.28)



```

1 %include "in_out.asm"
2
3 SECTION .data
4 msg db "Результат:", 0
5
6 SECTION .text
7 global _start
8
9 _start:
10 pop ecx
11 pop edx
12 sub ecx, 1
13 mov esi, 0
14 mov edi, 7
15
16 next:
17 cmp ecx, 0
18 jz _end
19 pop eax
20 call atoi

```

Рис. 4.28: Редактирование файла

Создаю исполняемый файл и запускаю его, чтобы проверить работу программы (рис. 4.29)

```

earepkina@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-task.asm
earepkina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-task lab8-task.o
earepkina@fedora:~/work/arch-pc/lab08$ ./lab8-task
Результат:0
earepkina@fedora:~/work/arch-pc/lab08$ ./lab8-task 1 46 7
Результат:399
earepkina@fedora:~/work/arch-pc/lab08$

```

Рис. 4.29: Запуск исполняемого файла

Программа работает верно.

Текст программы:

```
%include "in_out.asm"
```

```
SECTION .data
```

```
msg db "Результат:", 0
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
pop ecx
```

```
pop edx
```

```
sub ecx, 1
```

```
mov esi, 0
```

```
mov edi, 7
```

```
next:
```

```
cmp ecx, 0
```

```
jz _end
```

```
pop eax
```

```
call atoi
```

```
call _f
```

```
add esi, eax
```

```
loop next
```

```
_end:
```

```
mov eax, msg
```

```
call sprint
```

```

mov eax, esi
call iprintLF
call quit
_f:
inc eax
mul edi
ret

```

2. Создаю файл и ввожу туда текст программы из листинга 9.3 (рис. 4.30)



Рис. 4.30: Редактирование файла

Создаю исполняемый файл и запускаю его, чтобы проверить работу программы (рис. 4.31)

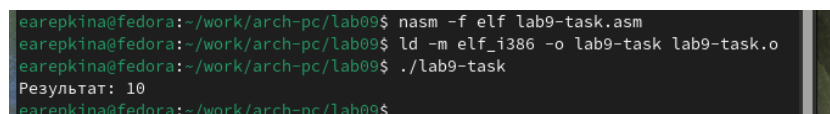


Рис. 4.31: Запуск исполняемого файла

Убеждаюсь, что программа работает неверно.

Создаю исполняемый файл для работы с GDB и запускаю его через режим отладки. Создаю брейкпойнт и пошагово просматриваю программу (рис. 4.32)

```

Register group: general
eax 0x804a000 134520832 ecx 0x4 4
edx 0x0 0 ebx 0xa 10
esp 0xffffc218 0xffffc218 ebp 0x0 0x0
esi 0x0 0 edi 0xa 10
eip 0x8049010 0x8049010 <sprint+1> eflags 0x206 [ PF IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

0x804900f <sprint> push %edx
> 0x8049010 <sprint+1> push %ecx
0x8049011 <sprint+2> push %ebx
0x8049012 <sprint+3> push %eax
0x8049013 <sprint+4> call 0x8049000 <len>
0x8049018 <sprint+9> mov %eax,%edx
0x804901a <sprint+11> pop %eax
0x804901b <sprint+12> mov %eax,%ecx
0x804901d <sprint+14> mov $0x1,%ebx
0x8049022 <sprint+19> mov $0x4,%eax
0x8049027 <sprint+24> int $0x80

native process 24231 In: sprint
(gdb) si
Breakpoint 3, _start () at test3.asm:10
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si

```

Рис. 4.32: Просмотр программы

Замечаю, что после выполнения инструкции `mul` программы умножит 4 на 2 на не на 5, как должно быть. Из-за этого программа выдает неверный результат. Далле открываю файл с программой и исправляю ошибку (рис. 4.33)

```

Открыть [v] [+]*lab9-task.asm Сохранить [≡] x
~/work/arch-pc/lab09

2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add eax, ebx
11 mov ecx,4
12 mul ecx
13 add eax,5
14 mov edi,eax
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
21

```

Рис. 4.33: Редактирование программы

Создаю исполняемый файл и запускаю его, чтобы проверить работу программы (рис. 4.34)


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

Рис. 4.34: Запуск исполняемого файла

Теперь программа работает верно.

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
```

5 Выводы

После выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и познакомилась с методами отладки при помощи GDB и его основными возможностями.

Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс,
- 11.
12. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
13. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
14. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
15. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-

- е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
16. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
 17. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер,
 18. — 1120 с. — (Классика Computer Science).