

Лабораторная работа №9

Дисциплина: Архитектура компьютера

Серебрякова Дарья Ильинична

Содержание

1	Цель работы	5
2	Задания	6
3	Теоретическое введение	7
3.1	Понятие об отладке	7
3.2	Методы отладки	8
4	Выполнение лабораторной работы	9
4.1	Реализация подпрограмм в NASM	9
4.2	Отладка программ с помощью GDB	12
4.3	Добавление точек останова	16
4.4	Работа с данными программы в GDB	17
4.5	Обработка аргументов командной строки в GDB	19
4.6	Задания для самостоятельной работы	21
5	Вывод	23
	Список литературы	24

Список иллюстраций

4.1	1	9
4.2	2	10
4.3	3	10
4.4	4	11
4.5	5	11
4.6	6	12
4.7	7	13
4.8	8	13
4.9	9	14
4.10	10	15
4.11	11	16
4.12	12	16
4.13	13	17
4.14	14	17
4.15	15	18
4.16	16	18
4.17	17	18
4.18	18	19
4.19	19	19
4.20	20	20
4.21	21	20
4.22	22	21
4.23	23	21
4.24	24	22

Список таблиц

1 Цель работы

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

2 Задания

1. Ознакомиться с понятием отладки
2. Ознакомиться со структурой подпрограмм
3. Научиться работать с подпрограммами и отладкой с помощью GDB

3 Теоретическое введение

3.1 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, 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 начнёт-

ся заново.

3.2 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:

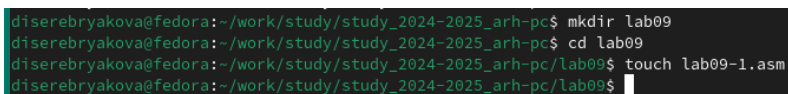
- Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом);
- Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, 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

4.1 Реализация подпрограмм в NASM

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



```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc$ mkdir lab09
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc$ cd lab09
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ touch lab09-1.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$
```

Рис. 4.1: 1

Изучаю текст программы из предложенного листинга и ввожу его в только что созданный файл (рис. 4.2).

```

/home/diserebryakova/work/study/study_2024-2025_arh-pc/lab09/1
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

```

Рис. 4.2: 2

Создаю исполняемый файл и проверяю его работу (рис. 4.3).

```

diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ nasm -f elf lab9-1.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ ld -m elf_i386 -o lab9-1 lab
9-1.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ ./lab9-1
Введите x: 2
2x+7=11
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$

```

Рис. 4.3: 3

Значение функции при введенном с клавиатуры x посчитано верно. Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x + 1$ (рис. 4.4).

```

GNU nano 7.2 /home/diserebryakova/work/study/study_2024-2025_arh-pc/lab
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2(3x-1)+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:
push eax
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
pop eax
ret ; выход из подпрограммы
_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret

```

Рис. 4.4: 4

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

```

diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ nasm -f elf lab9-1.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ ld -m elf_i386 -o lab9-1 lab
9-1.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ ./lab9-1
Введите x: 2
2(3x-1)+7=17
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$

```

Рис. 4.5: 5

Значение подсчитано верно, значит программа написана правильно

4.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm и ввожу в него текст программы из предложенного листинга (рис. 4.6).

```
/home/diserebryakova/work/study/study_2024-2025_arh-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
```

Рис. 4.6: 6

Создаю исполняемый файл, добавив отладочную информацию для работы с GDB. Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 4.7).

```

diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ nasm -f elf -g -l lab9-2.lst
: lab9-2.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ ld -m elf_i386 -o lab9-2 lab
9-2.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ gdb lab9-2
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 lab9-2...
(gdb) run
Starting program: /home/diserebryakova/work/study/study_2024-2025_arh-pc/lab09/lab9-2
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.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Download failed: Нет маршрута до узла. Continuing without separate debug info for system-sup
plied DSO at 0xf7ffc000.
Hello, world!
[Inferior 1 (process 3799) exited normally]
(gdb)

```

Рис. 4.7: 7

Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запускаю её командой `run` (рис. 4.8).

```

(gdb) break _start
Breakpoint 1 at 0x8040000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/diserebryakova/work/study/study_2024-2025_arh-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-sup
plied DSO at 0xf7ffc000.
Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4
(gdb)

```

Рис. 4.8: 8

Далее просматриваю дисассимилированный код программы с помощью ко-манды `disassemble` начиная с метки `_start` (рис. 4.9).

```

(gdb) run
Starting program: /home/diserebryakova/work/study/study_2024-2025_arh-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.

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

Переключаюсь на отображение команд с Intel'овским синтаксисом (рис. 4.10).

```

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

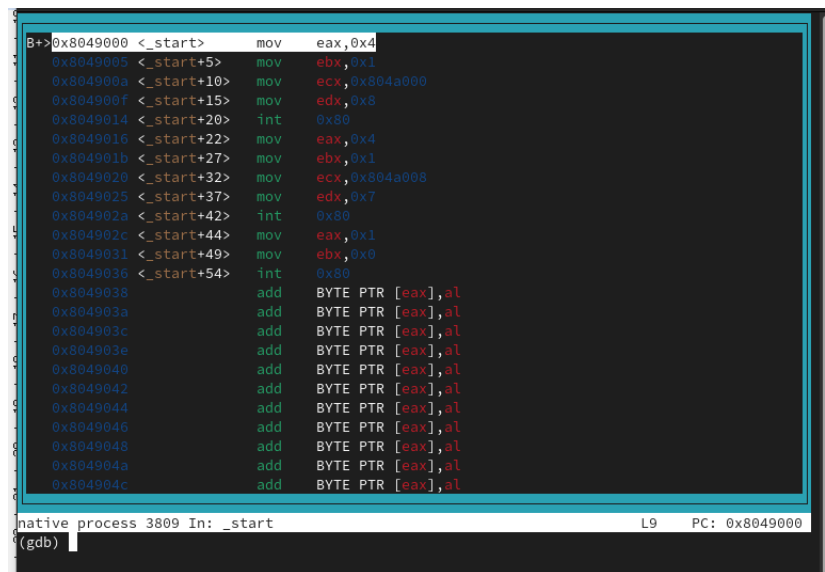
```

Рис. 4.10: 10

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).

Включаю режим псевдографики для более удобного анализа программы (рис.

4.11).

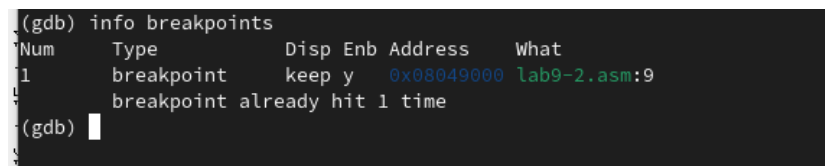


```
B> 0x8049000 <_start> mov eax, 0x4
0x8049005 <_start+5> mov ebx, 0x1
0x804900a <_start+10> mov ecx, 0x804a000
0x804900f <_start+15> mov edx, 0x8
0x8049014 <_start+20> int 0x80
0x8049016 <_start+22> mov eax, 0x4
0x804901b <_start+27> mov ebx, 0x1
0x8049020 <_start+32> mov ecx, 0x804a008
0x8049025 <_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
native process 3809 In: _start L9 PC: 0x8049000
(gdb)
```

Рис. 4.11: 11

4.3 Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверяю это с помощью команды `info breakpoints` (рис. 4.12).



```
(gdb) info breakpoints
Num    Type             Disp Enb Address            What
1      breakpoint      keep y   0x08049000 lab9-2.asm:9
breakpoint already hit 1 time
(gdb)
```

Рис. 4.12: 12

Устанавливаю еще одну точку останова по адресу предпоследней инструкции (`mov ebx, 0x0`) и просматриваю информацию о всех установленных точках останова (рис. 4.13).


```

(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab9-2.asm:9
        breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(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   0x08049031 lab9-2.asm:20
(gdb)

```

Рис. 4.13: 13

4.4 Работа с данными программы в GDB

Смотрю содержимое регистров с помощью команды info registers (рис. 4.14).

```

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

native process 4538 In: _start
eax            0x8                8
ecx            0x804a000          134520832
edx            0x8                8
ebx            0x1                1
esp            0xffffcfb0         0xffffcfb0
ebp            0x0                0x0
esi            0x0                0
edi            0x0                0
eip            0x8049016          0x8049016 <_start+22>
eflags         0x202              [ IF ]
cs             0x23               35
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.14: 14

Смотрю значение переменной msg1 по имени (рис. 4.15).

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

Рис. 4.15: 15

Смотрю значение переменной msg2 по адресу (рис. 4.16).

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

Рис. 4.16: 16

Меняю первый символ переменной msg1 и первый символ переменной msg2 (рис. 4.17).

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg2='x'
(gdb) x/1sb *msg2
'msg2' has unknown type; cast it to its declared type
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "xorld!\n\034"
(gdb) █
```

Рис. 4.17: 17

С помощью команды set изменяю значение регистра ebx (рис. 4.18).

```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) p/t $ebx
$2 = 110010
(gdb) p/s $ecx
$3 = 134520840
(gdb) p/x $ecx
$4 = 0x804a008
(gdb)

```

Рис. 4.18: 18

4.5 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 в файл с именем lab09-3.asm (рис. 4.19).

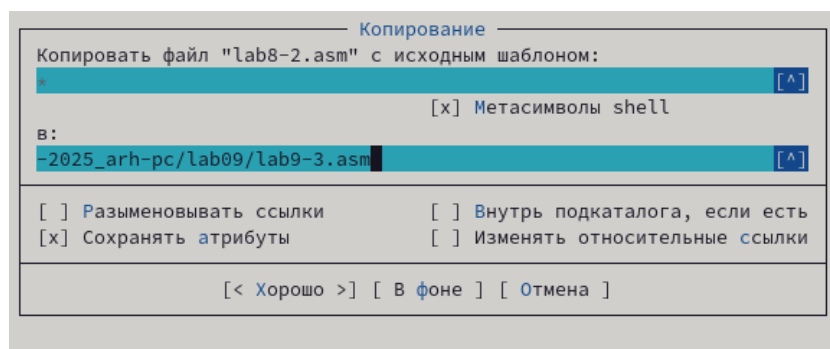


Рис. 4.19: 19

Создаю исполняемый файл и загружаю исполняемый файл в отладчик, указав

аргументы (рис. 4.20).

```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ nasm -f elf -g -l lab9-3.lst
lab9-3.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ ld -m elf_i386 -o lab9-3 lab
9-3.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ gdb --args lab9-3 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 lab9-3...
(gdb)
```

Рис. 4.20: 20

Исследую расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала устанавливаю точку останова перед первой инструкцией в программе и запускаю ее (рис. 4.21).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/diserebryakova/work/study/study_2024-2025_arh-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]) 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
Download failed: Нет маршрута до узла. Continuing without separate debug info for system-sup
plied DSO at 0xf7ffc000.

Breakpoint 1, _start () at lab9-3.asm:5
5      pop еск ; Извлекаем из стека в 'еск' количество
(gdb) x/x $esp
0xffffcf70: 0x00000005
(gdb)
```

Рис. 4.21: 21

Адрес вершины стека хранится в регистре есп и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

Просматриваю остальные позиции стека. По адресу [esp+4] располагается адрес в памяти где находится имя программы, по адресу [esp+8] хранится адрес первого

аргумента, по адресу [esp+12] – второго и т.д. (рис. 4.22).

```
(gdb) x/x $esp
0xffffcf70: 0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffd135: "/home/diserebryakova/work/study/study_2024-2025_arh-pc/lab09/lab9-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd179: "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd18b: "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd19c: "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd19e: "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.22: 22

4.6 Задания для самостоятельной работы

Создаю файл lab9-4.asm и ввожу в него программу из предложенного листинга (рис. 4.23).

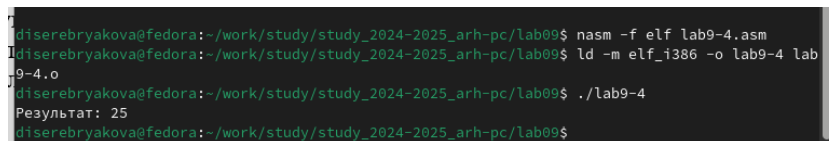
```
/home/diserebryakova/work/study/study_2024-2025_arh-pc
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 4.23: 23

Запускаю программу в режиме отладчика и пошагово через si 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 результат программы неверно подсчитывает функцию

Меняю код программы и запускаю ее повторно (рис. 4.24).



```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ nasm -f elf lab9-4.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ ld -m elf_i386 -o lab9-4 lab
9-4.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$ ./lab9-4
Результат: 25
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab09$
```

Рис. 4.24: 24

5 Вывод

В ходе выполнения работы приобретены навыки написания программ с использованием подпрограмм. Также ознакомилась с методами отладки при помощи GDB и его основными возможностями

Список литературы

Лабораторная работа №9