

Лабораторная работа №9. Понятие подпрограммы. Отладчик GDB

Простейший вариант

Диана Садова Алексеевна

Содержание

1	Цель работы	5
2	Задание	6
2.1	Порядок выполнения лабораторной работы	6
2.1.1	Реализация подпрограмм в NASM	6
2.1.2	Отладка программ с помощью GDB	11
2.1.3	Добавление точек останова	17
2.1.4	Работа с данными программы в GDB	18
2.1.5	Обработка аргументов командной строки в GDB	22
3	Теоретическое введение	25
4	Выполнение лабораторной работы	26
4.1	Задание для самостоятельной работы	27
4.1.1	Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму (рис. 4.1),(рис. 4.2).	27
4.1.2	В листинге 9.3 приведена программа вычисления выражения $(3 + 2) * 4 + 5$. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее (рис. 4.3),(рис. 4.4),(рис. 4.5),(рис. 4.6).	29
5	Выводы	32
	Список литературы	33

Список иллюстраций

2.1	Создаем каталог, файл и проверяем их наличие	6
2.2	Вводим код программы	7
2.3	Создаем исполняемый файл и проверяем его работу	8
2.4	Изменяем код программы	10
2.5	Проверяем его работу	11
2.6	Создаем файл и проверяем его наличие	11
2.7	Вводим программу	12
2.8	Транслируем программу и загружаем исполняемый файл в отладчик gdb	13
2.9	Запускаем работу программы	13
2.10	Устанавливаем брейкпоинт на метку <code>_start</code> и запускаем программу	14
2.11	Просматриваем дисассимилированный код, начиная с метки <code>_start</code>	14
2.12	Подключаем на отображение команд	15
2.13	Включаем режим псевдографики	16
2.14	Включаем режим псевдографики	16
2.15	Проверяем установилась ли точка на метку <code>_start</code>	17
2.16	Устанавливаем еще одну точку и смотрим по ней информацию . .	18
2.17	Выводим содержимое памяти	19
2.18	Как можно использовать <code>set</code>	20
2.19	Меняем значение регистра <code>ebx</code>	21
2.20	Завершаем выполнение программы	22
2.21	Выходим из GDB	22
2.22	Скопировали файл с новым названием	22
2.23	Создаем исполняемый файл	22
2.24	Загружаем исполняемый файл в отладчик	23
2.25	Устанавливаем точку остановки	23
2.26	Выводим вершину сетки	24
2.27	Просматриваем остальные позиции стеки	24
4.1	Преобразуем код программы	27
4.2	Запускаем код программы	28
4.3	Вводим код	29
4.4	Отладка GDB. Ищем проблему	30
4.5	Устраняем проблему	30
4.6	Запускаем программу	31

Список таблиц

1 Цель работы

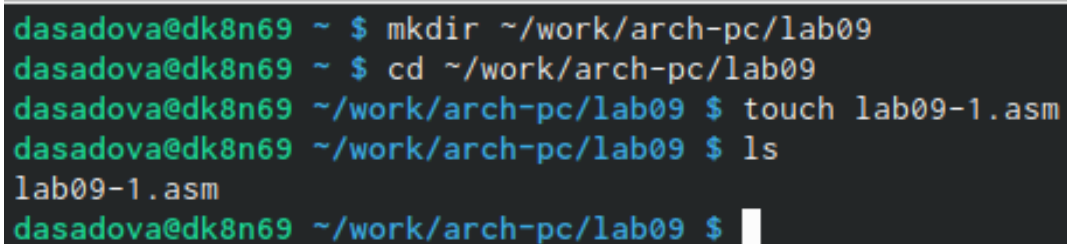
Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Задание

2.1 Порядок выполнения лабораторной работы

2.1.1 Реализация подпрограмм в NASM

2.1.1.1 Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и создайте файл lab09-1.asm: (рис. 2.1).



```
dasadova@dk8n69 ~ $ mkdir ~/work/arch-pc/lab09
dasadova@dk8n69 ~ $ cd ~/work/arch-pc/lab09
dasadova@dk8n69 ~/work/arch-pc/lab09 $ touch lab09-1.asm
dasadova@dk8n69 ~/work/arch-pc/lab09 $ ls
lab09-1.asm
dasadova@dk8n69 ~/work/arch-pc/lab09 $
```

Рис. 2.1: Создаем каталог, файл и проверяем их наличие

2.1.1.2 В качестве примера рассмотрим программу вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы `_calcul`. В данном примере `x` вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучите текст программы (Листинг 9.1).

Листинг 9.1. Пример программы с использованием вызова подпрограммы (рис. 2.2).

```

lab09-1.asm      [-M--] 27 L:[ 1+34 35/ 35] *(707 / 707b) <EOF>
#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 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. 2.2: Вводим код программы

Первые строки программы отвечают за вывод сообщения на экран (call sprint), чтение данных введенных с клавиатуры (call sread) и преобразования введенных данных из символьного вида в численный (call atoi).

```

mov eax, msg
call sprint
mov ecx, x

```

```
mov edx, 80
call sread
mov eax,x
call atoi
```

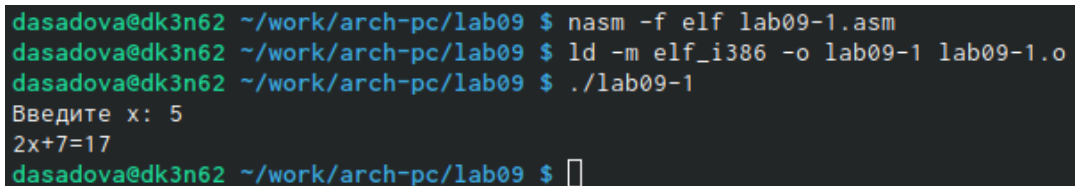
После следующей инструкции `call _calcul`, которая передает управление подпрограмме `_calcul`, будут выполнены инструкции подпрограммы:

```
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret
```

Инструкция `ret` является последней в подпрограмме и ее исполнение приводит к возвращению в основную программу к инструкции, следующей за инструкцией `call`, которая вызвала данную подпрограмму.

Последние строки программы реализуют вывод сообщения (`call sprint`), результата вычисления (`call iprintLF`) и завершение программы (`call quit`).

Введите в файл `lab09-1.asm` текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу.(рис. 2.3).



```
dasadova@dk3n62 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
dasadova@dk3n62 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
dasadova@dk3n62 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 5
2x+7=17
dasadova@dk3n62 ~/work/arch-pc/lab09 $
```

Рис. 2.3: Создаем исполняемый файл и проверяем его работу

Убедились, что код работает верно

Измените текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. Т.е. x передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение $g(x)$, результат возвращается в `_calcul` и

вычисляется выражение $f(g(x))$. Результат возвращается в основную программу для вывода результата на экран.(рис. 2.4),(рис. 2.5).

```

lab09-1.asm      [----] 32 L:[ 1+19 20/ 47] *(641 /1186b) 0096 0x060
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(g(x))=2(3x-1)+7 = ',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg ; вызов подпрограммы печати сообщения
call sprint ; 'Введите x: '
mov ecx, x
mov edx, 80
call sread ; вызов подпрограммы ввода сообщения
mov eax,x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, %eax=x
call _calcul ; Вызов подпрограммы _calcul
mov eax,result
call sprint
mov eax,[res]
call iprintLF

call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _subcalcul

mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы

; Подпрограмма вычисления
; выражения "3x-1"
_subcalcul:
mov ebx,3
mul ebx
sub eax,1
mov [res],eax
ret ; выход из подпрограммы

```

Рис. 2.4: Изменяем код программы

```

dasadova@dk3n65 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
dasadova@dk3n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
dasadova@dk3n65 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 5
f(g(x))=2(3x-1)+7 = 35
dasadova@dk3n65 ~/work/arch-pc/lab09 $

```

Рис. 2.5: Проверяем его работу

Код работает исправно, можно преступать к следующему пункту

2.1.2 Отладка программ с помощью GDB

Создайте файл lab09-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!):(рис. 2.6).

```

dasadova@dk3n62 ~/work/arch-pc/lab09 $ touch lab09-2.asm
dasadova@dk3n62 ~/work/arch-pc/lab09 $ ls
in_out.asm lab09-1 lab09-1.asm lab09-1.o lab09-2.asm
dasadova@dk3n62 ~/work/arch-pc/lab09 $

```

Рис. 2.6: Создаем файл и проверяем его наличие

Листинг 9.2. Программа вывода сообщения Hello world!(рис. 2.7).

```

lab09-2.asm      [-M--] 13 L:[ 1+15 16/ 21] *(236 / 293b) 0010 0x00A
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. 2.7: Вводим программу

Получите исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’.

Загрузите исполняемый файл в отладчик gdb:(рис. 2.8).

```

dasadova@dk3n62 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
dasadova@dk3n62 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
dasadova@dk3n62 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 12.1 vanilla) 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.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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...

```

Рис. 2.8: Транслируем программу и загружаем исполняемый файл в отладчик gdb

Проверьте работу программы, запустив ее в оболочке GDB с помощью команды `run` (сокращённо `r`):(рис. 2.9).

```

GNU gdb (Gentoo 12.1 vanilla) 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.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/a/dasadova/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 4446) exited normally]
(gdb)

```

Рис. 2.9: Запускаем работу программы

Для более подробного анализа программы установите брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и

запустите её.(рис. 2.10).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/a/dasadova/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
█
```

Рис. 2.10: Устанавливаем брейкпоинт на метку `_start` и запускаем программу

Посмотрите дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`(рис. 2.11).

```
(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.
```

Рис. 2.11: Просматриваем дисассимилированный код, начиная с метки `_start`

Переключитесь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`(рис. 2.12).

```

(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. 2.12: Подключаем на отображение команд

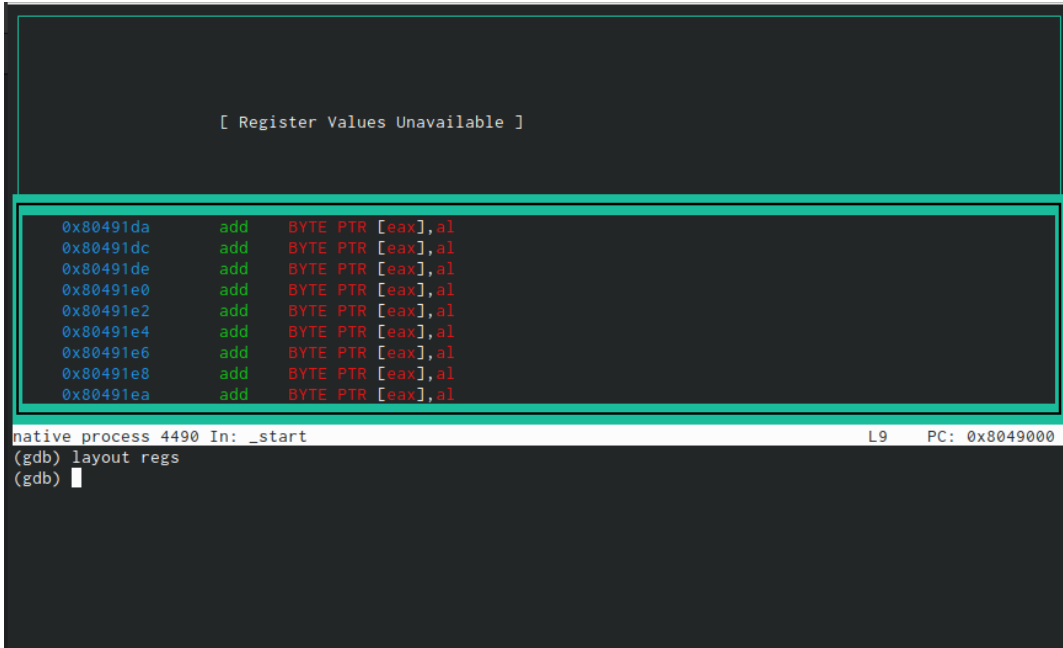
Перечислите различия отображения синтаксиса машинных команд в режимах АТТ и Intel.

1. Непосредственные операнды АТ&Т пишутся после \$ непосредственные операнды Intel не выделяются.
2. Регистровые операнды АТ&Т пишутся после %; регистровые операнды Intel не выделяются.
3. Абсолютные операнды АТ&Т jump/call пишутся после *; они не выделяются в синтаксисе Intel.

Включите режим псевдографики для более удобного анализа программы (рис. 2.13),(рис. 2.14):

```
End of assembler dump.  
(gdb) layout asm
```

Рис. 2.13: Включаем режим псевдографики



The screenshot shows the GDB interface with the 'layout asm' command executed. The window is divided into three main sections. The top section, titled '[Register Values Unavailable]', is currently empty. The middle section displays a list of assembly instructions with their addresses and operands, all of which are 'add BYTE PTR [eax], al'. The bottom section shows the GDB prompt and the command 'layout regs'.

Address	Instruction
0x80491da	add BYTE PTR [eax], al
0x80491dc	add BYTE PTR [eax], al
0x80491de	add BYTE PTR [eax], al
0x80491e0	add BYTE PTR [eax], al
0x80491e2	add BYTE PTR [eax], al
0x80491e4	add BYTE PTR [eax], al
0x80491e6	add BYTE PTR [eax], al
0x80491e8	add BYTE PTR [eax], al
0x80491ea	add BYTE PTR [eax], al

native process 4490 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb)

Рис. 2.14: Включаем режим псевдографики

В этом режиме есть три окна:

- В верхней части видны названия регистров и их текущие значения;
- В средней части виден результат дисассимилирования программы;
- Нижняя часть доступна для ввода команд.

2.1.3 Добавление точек останова

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать или как номер строки программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»:

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверьте это с помощью команды `info breakpoints` (кратко `i b`) (рис. 2.15).

```
(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
          breakpoint already hit 1 time
2        breakpoint       keep y  0x08049031 lab09-2.asm:20
(gdb) █
```

Рис. 2.15: Проверяем установилась ли точка на метку `_start`

Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции.

Определите адрес предпоследней инструкции (`mov ebx,0x0`) и установите точку останова (рис. 2.16).

Посмотрите информацию о всех установленных точках останова (рис. 2.16).

```

(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
          breakpoint already hit 1 time
2        breakpoint     keep y   0x08049031 lab09-2.asm:20
(gdb)

```

Рис. 2.16: Устанавливаем еще одну точку и смотрим по ней информацию

2.1.4 Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных.

Выполните 5 инструкций с помощью команды `stepi` (или `si`) и проследите за изменением значений регистров. Значения каких регистров изменяются?

Ответ: регистров `eax`, `ebx`, `ecx`, `edx`

Посмотреть содержимое регистров также можно с помощью команды `info registers` (или `i r`).

Для отображения содержимого памяти можно использовать команду `x` “<”адрес”>“, которая выдаёт содержимое ячейки памяти по указанному адресу. Формат, в котором выводятся данные, можно задать после имени команды через косую черту: `x/NFU` “<адрес”>“.

С помощью команды `x` & “<”имя переменной”>” также можно посмотреть содержимое переменной.

Посмотрите значение переменной `msg1` по имени.

```
(gdb) x/1sb &msg1
```

```
0x804a000 < msg1 >: "Hello,"
```

Посмотрите значение переменной `msg2` по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрите инструкцию `mov ecx,msg2` которая записывает в регистр `ecx` адрес переменной `msg2` (рис. 2.17).

```
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

native process 3929 In: _start
gs          0x0          0
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) █
```

Рис. 2.17: Выводим содержимое памяти

Изменить значение для регистра или ячейки памяти можно с помощью команды `set`, задав ей в качестве аргумента имя регистра или адрес. При этом перед именем регистра ставится префикс `$`, а перед адресом нужно указать в фигурных скобках тип данных (размер сохраняемого значения; в качестве типа данных можно использовать типы языка Си). Измените первый символ переменной `msg1` (рис. 2.18).

```

(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:          "hh1lo, "
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:          "Lor d!\n\034"
(gdb) 

```

Рис. 2.18: Как можно использовать set

Замените любой символ во второй переменной msg2.

Чтобы посмотреть значения регистров используется команда print /F (перед именем регистра обязательно ставится префикс \$):

p/F \$ "<"регистр">"

Выведете в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.

С помощью команды set измените значение регистра ebx (рис. 2.19).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) p/t $ebx
$2 = 110010
(gdb) set $ebx=2
(gdb) p/s $ebx
$3 = 2
(gdb) 
```

Рис. 2.19: Меняем значение регистра ebx

Объясните разницу вывода команд p/s \$ebx.

В p/s \$ebx = '2' мы вводим значение в шестнадцатеричном формате и у нас выходит \$1 = 50

Но p/s \$ebx = 2 мы вводим как значение в символьном виде, у нас идет перезапись и выходит \$3 = 2

Завершите выполнение программы с помощью команды continue (сокращенно c) или stepi (сокращенно si) и выйдите из GDB с помощью команды quit (сокращенно q) (рис. 2.20), (рис. 2.21).

```
(gdb) c
Continuing.
world!
[Inferior 1 (process 4393) exited normally]
(gdb) █
```

Рис. 2.20: Завершаем выполнение программы

```
[Inferior 1 (process 4393) exited normally]
(gdb) quit █
```

Рис. 2.21: Выходим из GDB

2.1.5 Обработка аргументов командной строки в GDB

Скопируйте 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. 2.22).

```
dasadova@dk3n65 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
dasadova@dk3n65 ~/work/arch-pc/lab09 $ ls
in_out.asm lab09-1 lab09-1.asm lab09-1.o lab09-2 lab09-2.asm lab09-2.lst lab09-2.o lab09-3.asm
dasadova@dk3n65 ~/work/arch-pc/lab09 $ █
```

Рис. 2.22: Скопировали файл с новым названием

Создайте исполняемый файл (рис. 2.23).

```
dasadova@dk3n65 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
dasadova@dk3n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
dasadova@dk3n65 ~/work/arch-pc/lab09 $ ls
in_out.asm lab09-1.asm lab09-2 lab09-2.lst lab09-3 lab09-3.lst
lab09-1 lab09-1.o lab09-2.asm lab09-2.o lab09-3.asm lab09-3.o
dasadova@dk3n65 ~/work/arch-pc/lab09 $ █
```

Рис. 2.23: Создаем исполняемый файл

Для загрузки в gdb программы с аргументами необходимо использовать ключ `-args`. Загрузите исполняемый файл в отладчик, указав аргументы (рис. 2.24).

```
dasadova@dk3n65 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Gentoo 12.1 vanilla) 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.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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. 2.24: Загружаем исполняемый файл в отладчик

Как отмечалось в предыдущей лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Исследуем расположение аргументов командной строки в стеке после запуска программы с помощью gdb.

Для начала установим точку останова перед первой инструкцией в программе и запустим ее (рис. 2.25).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 8.
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/a/dasadova/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2
аргумент\ 3

Breakpoint 1, _start () at lab09-3.asm:8
8      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) █
```

Рис. 2.25: Устанавливаем точку остановки

Адрес вершины стека храниться в регистре `esp` и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы) (рис. 2.26).

```
(gdb) x/x $esp
0xffffc2e0:      0x00000005
(gdb)
```

Рис. 2.26: Выводи вершину сетки

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

Посмотрите 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. 2.27).

```
(gdb) x/s *(void**)(esp + 4)
0xffffc579:      "/afs/.dk.sci.pfu.edu.ru/home/d/a/dasadova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffc5be:      "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffc5d0:      "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffc5e1:      "2"
(gdb) x/s *(void**)(esp + 20)
A syntax error in expression, near `'.
(gdb) x/s *(void**)(esp + 20)
0xffffc5e3:      "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:      <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.27: Просматриваем остальные позиции стеки

Объясните, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.

Ответ:потому что размер слова (или размер указателя) составляет 4 байта. И так как мы обращаемся к следующему элементу, наше значение всегда увеличивается на это 4 бвйта.

3 Теоретическое введение

4 Выполнение лабораторной работы

4.1 Задание для самостоятельной работы

4.1.1 Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму (рис. 4.1),(рис. 4.2).

```
lab8-zadanie.asm  [----] 35 L:[ 1+14 15/ 40] *(573 /1499b) 0041 0x029
%include 'in_out.asm'
SECTION .data
msg1 db "Функция: f(x)=8*x-3",0
msg db "Результат: ",0
SECTION .text
global _start
_start:
mov eax,msg1
call sprintf
pop ecx ; Извлекаем из стека в 'ecx' количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в 'edx' имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
; аргументов без названия программы)
mov esi, 0 ; Используем 'esi' для хранения
; промежуточных сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку '_end')
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
call _calcul
; след. аргумент 'esi=esi+eax'
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprintf
mov eax, esi ; записываем сумму в регистр 'eax'
call iprintLF ; печать результата
call quit ; завершение программы

_calcul:
mov ebx,8
mul ebx
sub eax,3
add esi,eax
mov esi,eax
ret
```

Рис. 4.1: Преобразуем код программы

```
dasadova@dk4n71 ~/work/arch-pc/lab09 $ ./lab8-zadanie 1 2 3
Функция:  $f(x)=8*x-3$ 
Результат: 21
dasadova@dk4n71 ~/work/arch-pc/lab09 $
```

Рис. 4.2: Запускаем код программы

После запуска программа работает исправно, значит можно переходить к следующему заданию

4.1.2 В листинге 9.3 приведена программа вычисления выражения $(3 + 2) * 4 + 5$. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее (рис. 4.3),(рис. 4.4),(рис. 4.5),(рис. 4.6).

```
lab09-zadanie.asm  [----]  9 L:[ 1+19 20/ 20] *(348 / 348b) <EOF>
%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. 4.3: Вводим код

```

Register group: general
eax 0x8 8 ecx 0x4 4
edx 0x0 0 ebx 0x5 5
esp 0xffffc330 0xffffc330 ebp 0x0 0x0
esi 0x0 0 edi 0x0 0
eip 0x80490fb 0x80490fb <_start+19> eflags 0x202 [ IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

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>

native process 5527 In: _start L13 PC: 0x80490fb
Breakpoint 1, _start () at lab09-zadanie.asm:8
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x80490e8 lab09-zadanie.asm:8
breakpoint already hit 1 time
(gdb) s
(gdb) s
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 4.4: Отладка GDB. Ищем проблему

```

lab09-zadanie.asm [----] 10 L: [ 1+13 14/ 20] *(232 / 348b) 0120 0x078
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov eax,3
mov ebx,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

```

Рис. 4.5: Устраняем проблему

```
dasadova@dk4n71 ~/work/arch-pc/lab09 $ nasm -f elf lab09-zadanie.asm
dasadova@dk4n71 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-zadanie lab09-zadanie.o
dasadova@dk4n71 ~/work/arch-pc/lab09 $ ./lab09-zadanie
Результат: 25
dasadova@dk4n71 ~/work/arch-pc/lab09 $
```

Рис. 4.6: Запускаем программу

Создаем файл с именем `lab09-zadanie.asm` - это файл с нашей дальнейшей работой. В `lab09-zadanie.asm` вводим код программы. Я убедилась, что в нем есть ошибка, когда запустила в первый раз, будем исправлять. Заходим в отладчик GDB и смотрим в какой именно момент случается проблема с значениями, проверяем помощью операции `si`. Поняли, что ошибка случается при умножении. Переходим в код программы и устраняем ошибку. Запускаем исполняемый файл и убеждаемся, что проблема устранена.

5 Выводы

Приобрели навыки написания программ с использованием подпрограмм. Познакомились с методами отладки при помощи GDB и его основными возможностями.

Список литературы