Отчётпо лабораторной работе №9

Дисциплина: Архитектура компьютера

Максимова Дарья Валерьевна

Содержание

Список иллюстраций

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. Реализация подпрограмм в NASM
2. Отладка программам 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 Выполнение лабораторной работы

3.1 Реализация подпрограмм в NASM

По аналогии с предыдущими лабораторными работами, файлы asm я буду создавать в каталоге “work/arch-pc”. В файле lab9-1.asm вписываю программу из листинга 9.1, которую я внимательно изучила. Создаю исполняемый файл и запускаю его в работу. (рис. 3.1).

```
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ ls
in_out.asm  lab9-1.asm
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ nasm -f elf lab9-1.asm
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-1 lab9-1.o
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ ./lab9-1
Введите x: 2
2x+7=11
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $
```

Рис. 3.1: Первые действия в лабораторной работе и результат программы

После этого я переписываю программу согласно методическому материалу (рис. 3.2).

```
%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:
call _subcalcul
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret
_subcalcul:
mov ebx,3
mul ebx
sub eax,1
mov [res],eax

ret ; выход из подпрограммы
```

Рис. 3.2: Текст программы

Создаю исполняемый файл и убеждаюсь в том, что все работает успешно.(рис. 3.3).

```
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ nasm -f elf lab9-1.asm
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-1 lab9-1.o
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ ./lab9-1
Введите x: 2
2(3x-1)+7=17
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ ./lab9-1
Введите x: 3
2(3x-1)+7=23
```

Рис. 3.3: Результат программы

3.2 Отладка программам с помощью GDB

В новом файле lab9-2.asm записываю текст программы из листинга 9.2. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ провожу с ключом '-g'. И загружаю файл в откладчик gbd, а затем проверяю выполнение программы с помощью команды run (рис. 3.4).

```
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-2 lab9-2.o
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ gdb lab9-2
GNU gdb (Gentoo 14.2 vanilla) 14.2
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-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 lab9-2...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/v/dvmaksimova/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 4723) exited normally]
```

Рис. 3.4: Работа программы

Для более подробного анализа программы устанавливаю брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запускаю её. (рис. 3.5).

```
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 lab9-2...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/v/dvmaksimova/work/arch-pc/lab0
9/lab9-2
Hello, world!
[Inferior 1 (process 4723) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/v/dvmaksimova/work/arch-pc/lab0
9/lab9-2

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

Рис. 3.5: Брейкпоинт

Просматриваю дисассимилированный код программы с помощью команды disassemble начиная с метки _start (рис. 3.6).

```
(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. 3.6: Дисассимилированный код программы

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel (рис. 3.7).

```
(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. 3.7: Отображение команд

Есть некоторые различия в отображениях в этих режимах, а именно в виде колонки с текстом программы: в Intel'е она выглядит, как *"$0x{операнд},%{регистер}"*, а в ATT - *"{регистер},0x{операнд}"*

Включаю режим псевдографики для удобного анализа программ (рис. 3.8).

```
    [ 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

native process 4847 In: _start                          L9    PC: 0x8049000
(gdb) layout regs
(gdb)
```

Рис. 3.8: Псевдографика.Режим

3.3 Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (_start). Проверяю это с помощью команды info breakpoints (рис. 3.9).

Рис. 3.9: Информация о точке остановы

Я установила еще одну точку останова по адресу инструкции, и теперь когда мы запрашиваем информацию о брейкпоинтах, нам показываются две точки останова: (рис. 3.10).

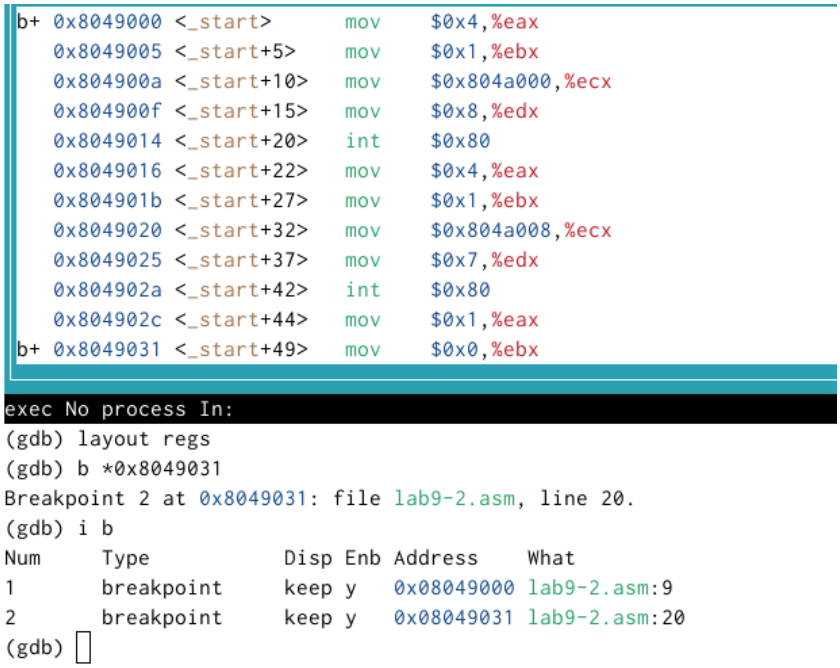

Рис. 3.10: Точка останова

3.4 Работа с данными программы в GDB

Для того, что бы посмотреть значения регистров можно воспользоваться командой “i r”. Вот какие значения регистров вывелись (рис. 3.11).

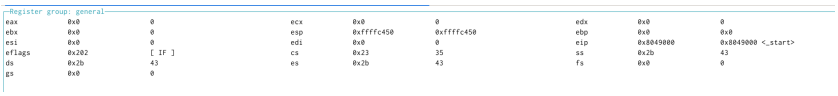

Рис. 3.11: Значения регистров

С помощью команды x & также можно посмотреть содержимое переменной. Я посмотрела значение переменной msg1 по имени и вот результат. (рис. 3.12).

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

Рис. 3.12: Просмотр значения переменной

Аналогично можно посмотреть значение переменной msg2 по адресу. (рис. 3.13).

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

Рис. 3.13: Значение другой переменной

Теперь я изменяю первый символ переменной msg1 (рис. 3.14).

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

Рис. 3.14: msg1=hello,

А также я изменила разные символы в переменной msg2 (рис. 3.15).

```
(gdb) set {char}0x804a00d='a'
(gdb) x/1sb &msg2
0x804a008 <msg2>:       "Morlda\n\034"
(gdb)
```

Рис. 3.15: Измененные символы

Теперь изменим значение регистра ebx: (рис. 3.16).

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

Рис. 3.16: Изменение ebx

Команда set позволяе изменить значение регистра, проделываем это с регистром ebx, и использую команду p/s, чтоб промотреть значение.

3.5 Обработка аргументов командной строки в GDB

Создаю исполняемый файл из файла, с которым я работала в процессе выполнения лбораторной работы №8.

А затем загружаю данный файл в отладчик, предварительно указав ключ к работе с аргументами, и указываю эти самые аргументы (рис. 3.17).

```
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 la
b09-3.o
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1
 аргумент 2 'аргумент 3'
GNU gdb (Gentoo 14.2 vanilla) 14.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gp
l.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) b _start
```

Рис. 3.17: Переход к отладчику

Для начала устанавливаем первую точку останова, а затем запускаем программу (рис. 3.18).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/v/dvmaksimova/work/ar
ch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab09-3.asm:5
5       pop ecx ; Извлекаем из стека в `ecx` количество
```

Рис. 3.18: b-point + run

На данном этапе, мы можем просмотреть позиции стека отдельно по адресам (рис. 3.19).

```
(gdb) x/x $esp
0xffffc400:     0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffc659:     "/afs/.dk.sci.pfu.edu.ru/home/d/v/dvmaksimova/work/arc
h-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffc6a1:     "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffc6b3:     "аргумент"
(gdb) x/s *(void**)($esp + 20)
0xffffc6c6:     "аргумент 3"
(gdb)
```

Рис. 3.19: Позиции стека

3.6 Задание для самостоятельной работы

Переходим к выполнению заданий самостоятельного характера.

№1

Я преобразовываю программу , с которой я работала в процесее 8 лабораторной работы. Я должна реализовать вычисление функции в подпрограмме (до этого вычисления были в основной программе).

Вот таким образом выглядит код программы: (рис. 3.20).

```
%include 'in_out.asm'
SECTION .data
msg1 db "Функция: f(x)=3x-1", 0
msg2 db "Результат: ",0
SECTION .text
global _start
_start:

pop ecx
pop edx
sub ecx,1
mov esi, 0

next:
cmp ecx,0h
jz _end
pop eax
call atoi

call mom

add esi,eax
loop next
_end:
mov eax, msg1
call sprintLF
mov eax, msg2
call sprint
mov eax, esi
call iprintLF
call quit

mom:
mov ebx, eax
add eax, ebx
add eax, ebx
sub eax,1
ret

```

Рис. 3.20: Код программы

Теперь проверим как работает программа: (рис. 3.21).

```
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ ./nomer 2
Функция: f(x)=3x-1
Результат: 5
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ ./nomer 7
Функция: f(x)=3x-1
Результат: 20
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $
```

Рис. 3.21: Результат

№2

При запуске программа вычисления выражения (3 + 2) 4 + 5 дает неверный результат. Программа должна выводить 25, а выводит в ответе 10.(рис. 3.22).

```
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o nomer2 nomer2.o
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ ./nomer2
Результат: 10
```

Рис. 3.22: Ложный результат(символьный)

Моя задача - определить ошибку с помощью отладчика. Ошибка заключается в том, что регистр eax был умножен на 4, хотя для правильности нам надо было умножать ebx

Саомстоятельно постараюсь исправить код. Вот так выглядит код с внесением моих изменений (рис. 3.23).

```
1 %include 'in_out.asm'
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
```

Рис. 3.23: Код

Теперь я проверяю работу кода, и радуюсь, что все нормально работает!!!:) (рис. 3.24).

```
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ nasm -f elf nomer2.asm
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o nomer2 nomer2.o
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $ ./nomer2
Результат: 25
dvmaksimova@dk8n80 ~/work/arch-pc/lab09 $
```

Рис. 3.24: Результат

4 Выводы

Я приобрела навыки написания программ с использованием подпрограмм, познакомилась с методами отладки при помощи GDB и его основными возможностями.