

Лабораторная работа №10

Дисциплина: Архитектура компьютера

Кондратьев Арсений Вячеславович

07.10.2022

Содержание

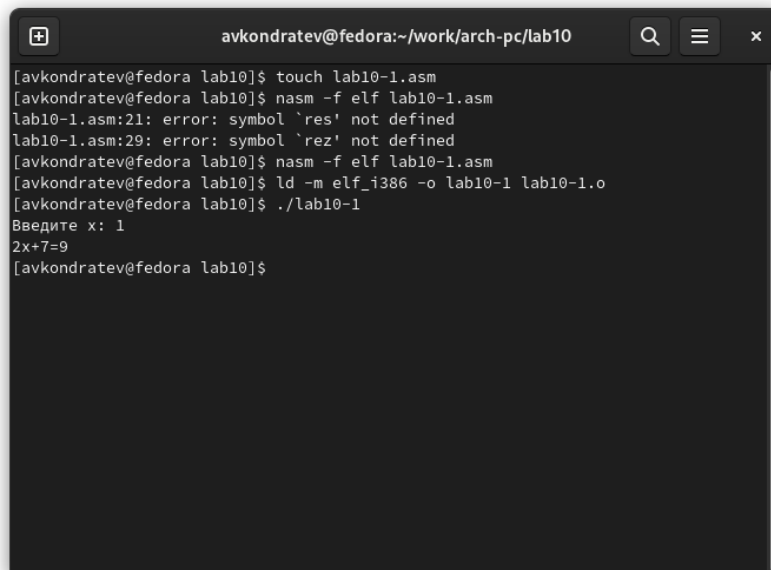
1	Цель работы	3
2	Выполнение лабораторной работы	4
3	Выводы	11
4	Контрольные вопросы	12

1 Цель работы

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

2 Выполнение лабораторной работы

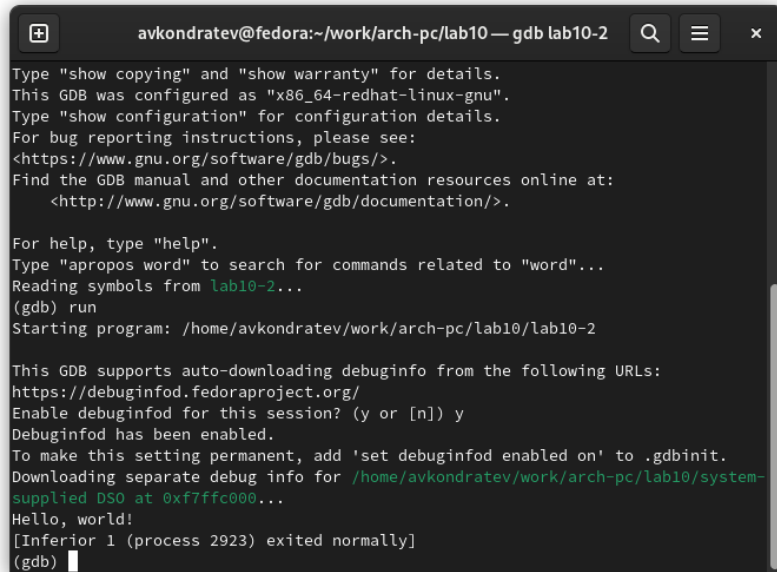
1. Реализовал пример программы с использованием вызова подпрограммы(рис.2.1)



```
avkondratev@fedora:~/work/arch-pc/lab10
[avkondratev@fedora lab10]$ touch lab10-1.asm
[avkondratev@fedora lab10]$ nasm -f elf lab10-1.asm
lab10-1.asm:21: error: symbol `res' not defined
lab10-1.asm:29: error: symbol `rez' not defined
[avkondratev@fedora lab10]$ nasm -f elf lab10-1.asm
[avkondratev@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[avkondratev@fedora lab10]$ ./lab10-1
Введите x: 1
2x+7=9
[avkondratev@fedora lab10]$
```

Figure 2.1: Рис. 1

2. Реализовал программу вывода сообщения Hello world! и проверил работу программы, запустив ее в оболочке GDB с помощью команды run (рис.2.2)



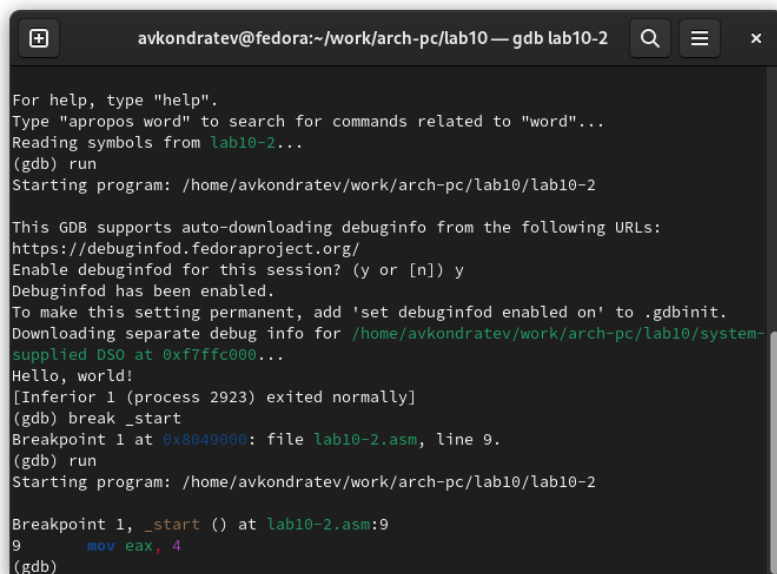
```
avkondratev@fedora:~/work/arch-pc/lab10 — gdb lab10-2
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
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 lab10-2...
(gdb) run
Starting program: /home/avkondratev/work/arch-pc/lab10/lab10-2

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 /home/avkondratev/work/arch-pc/lab10/system-
supplied DSO at 0xf7ffc000...
Hello, world!
[Inferior 1 (process 2923) exited normally]
(gdb)
```

Figure 2.2: Рис. 2

3. Установил брейкпоинт на метку _start (рис.2.3)



```
avkondratev@fedora:~/work/arch-pc/lab10 — gdb lab10-2
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab10-2...
(gdb) run
Starting program: /home/avkondratev/work/arch-pc/lab10/lab10-2

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 /home/avkondratev/work/arch-pc/lab10/system-
supplied DSO at 0xf7ffc000...
Hello, world!
[Inferior 1 (process 2923) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 9.
(gdb) run
Starting program: /home/avkondratev/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:9
9      mov eax, 4
(gdb)
```

Figure 2.3: Рис. 3

4. Посмотрел дисассимилированный код программы(рис.2.4)

```
avkondratev@fedora:~/work/arch-pc/lab10 — gdb lab10-2
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 9.
(gdb) run
Starting program: /home/avkondratev/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.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)
```

Figure 2.4: Рис. 4

5. Переключился на отображение команд с Intel'овским синтаксисом(рис.2.5)

```
avkondratev@fedora:~/work/arch-pc/lab10 — gdb lab10-2
[ Register Values Unavailable ]

B> 0x8049000 <_start>   mov     eax,0x4
    0x8049005 <_start+5> mov     ebx,0x1
    0x804900a <_start+10> mov     ecx,0x804a000
    0x804900f <_start+15> mov     edx,0x8
    0x8049014 <_start+20> int     0x80
    0x8049016 <_start+22> mov     eax,0x4
    0x804901b <_start+27> mov     ebx,0x1

native process 2960 In: _start          L9      PC: 0x8049000
(gdb) layout regs
(gdb)
```

Figure 2.5: Рис. 5

6. Посмотрел значение переменной msg2 по адресу и изменил значение ячейки памяти(рис.2.6)

```
avkondratev@fedora:~/work/arch-pc/lab10 — gdb lab10-2
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
b+ 0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80

native process 3453 In: _start L9 PC: 0x8049000
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab10-2.asm, line 20.
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb)
```

Figure 2.6: Рис. 6

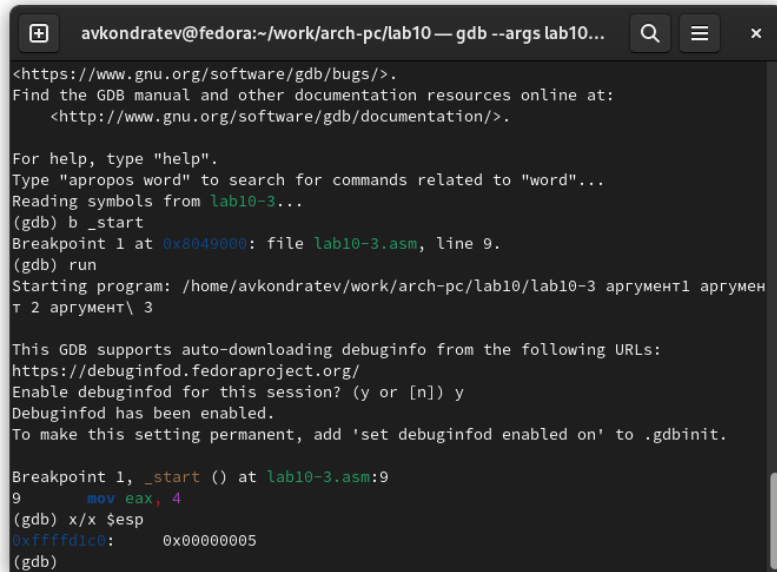
7. Заменял символ во второй переменной msg2(рис.2.7)

```
avkondratev@fedora:~/work/arch-pc/lab10 — gdb lab10-2
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

native process 3453 In: _start L9 PC: 0x8049000
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) x/1sb &msg2
0x804a008 <msg2>: "world!\n\034"
(gdb) set {char}0x804a008='G'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "Gorld!\n\034"
(gdb)
```

Figure 2.7: Рис. 7

8. Посмотрел количество аргументов командной строки(рис.2.8)



```
avkondratev@fedora:~/work/arch-pc/lab10 — gdb --args lab10...
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

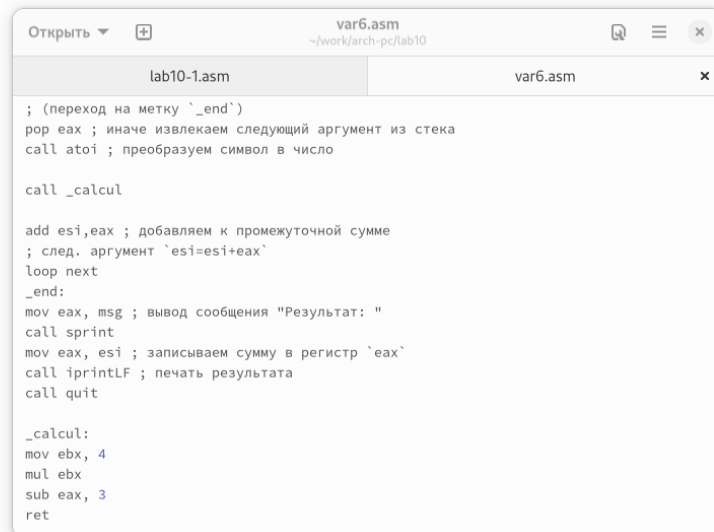
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab10-3...
(gdb) b _start
Breakpoint 1 at 0x8049000: file lab10-3.asm, line 9.
(gdb) run
Starting program: /home/avkondratev/work/arch-pc/lab10/lab10-3 аргумент1 аргумент 2 аргумент\ 3

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.

Breakpoint 1, _start () at lab10-3.asm:9
9      mov eax, 4
(gdb) x/x $esp
0xffffd1c0:  0x00000005
(gdb)
```

Figure 2.8: Рис. 8

9. Преобразовал программу из лабораторной работы №9, реализовав вычисление значения функции как подпрограмму(рис.2.9)(рис.2.10)



```
lab10-1.asm
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число

call _calcul

add esi, eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintf ; печать результата
call quit

_calcul:
mov ebx, 4
mul ebx
sub eax, 3
ret
```

Figure 2.9: Рис. 9


```
avkondratev@fedora:~/work/arch-pc/lab10
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.

Breakpoint 1, _start () at lab10-3.asm:9
9      mov eax, 4
(gdb) x/x $esp
0xffffd1c0:    0x00000005
(gdb) q
A debugging session is active.

    Inferior 1 [process 3625] will be killed.

Quit anyway? (y or n) y
[avkondratev@fedora lab10]$ nasm -f elf var6.asm
[avkondratev@fedora lab10]$ ld -m elf_i386 -o var6 var6.o
[avkondratev@fedora lab10]$ ./var6
Результат: 0
[avkondratev@fedora lab10]$ ./var6 1 2
Результат: 6
[avkondratev@fedora lab10]$
```

Figure 2.10: Рис. 10

10. С помощью отладчика GDB, анализируя изменения значений регистров, определил ошибку и исправил ее(рис.2.11)(рис.2.12)(рис.2.13)

```
avkondratev@fedora:~/work/arch-pc/lab10 — gdb self10-3
Register group: general
eax      0x8      8
ecx      [ Regist0x4Values Unavailabl4 ]
edx      0x0      0
ebx      0xa      10
esp      0xffffd200 0xffffd200

0x80490f4 <_start+12> mov    ecx,0x4
B+ 0x80490f9 <_start+17> mul    ecx
0x80490fb <_start+19> add    ebx,0x5
B+> 0x80490fe <_start+22> mov    edi,ebx
0x8049108 <_start+24> mov    eax,0x804a000
0x8049105 <_start+29> call   0x804900f <sprint>
0x804910a <_start+34> mov    eax,edi

exec No process In: L?? PC: ??
native process 4453 In: _start L?? PC: 0x80490fe

Breakpoint 1, 0x80490f9 in _start ()
(gdb) next
Single stepping until exit from function _start,
which has no line number information.

Breakpoint 2, 0x80490fe in _start ()
(gdb)
```

Figure 2.11: Рис. 11

```
avkondratev@fedora:~/work/arch-pc/lab10
Reading symbols from self10-3...
(No debugging symbols found in self10-3)
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
0x080490e8 <+0>:    mov     ebx,0x3
0x080490ed <+5>:    mov     eax,0x2
0x080490f2 <+10>:   add     ebx,eax
0x080490f4 <+12>:   mov     ecx,0x4
0x080490f9 <+17>:   mul     ecx
0x080490fb <+19>:   add     ebx,0x5
0x080490fe <+22>:   mov     edi,ebx
0x08049100 <+24>:   mov     eax,0x804a000
0x08049105 <+29>:   call   0x804900f <sprint>
0x0804910a <+34>:   mov     eax,edi
0x0804910c <+36>:   call   0x8049086 <iprintf>
0x08049111 <+41>:   call   0x80490db <quit>
End of assembler dump.
(gdb) layout asm
[avkondratev@fedora lab10]$ nasm -f elf self10-3.asm
[avkondratev@fedora lab10]$ ld -m elf_i386 -o self10-3 self10-3.o
[avkondratev@fedora lab10]$ ./self10-3
Результат: 25
[avkondratev@fedora lab10]$
```

Figure 2.12: Рис. 12

```
self10-3.asm
~/.work/arch-pc/lab10

self10-3.asm x var6.asm

%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov eax,2
add eax,3
mov ebx,4
mul ebx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintfLF
call quit
```

Figure 2.13: Рис. 13

3 Выводы

Я приобрел навыки написания программ с использованием подпрограмм. Познакомился с методами отладки при помощи GDB и его основными возможностями

4 Контрольные вопросы

1. Инструкция `call` и инструкция `ret`
2. С помощью инструкции `call` и имени подпрограммы
3. Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `esp` адрес соответствующей подпрограммы. Подпрограмма завершается инструкцией `ret`, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией `call`, и заносит его в `esp`.
4. Операнд необязателен, но если он есть, то после того, как будет считан адрес возврата, из стека будет удалено столько байтов, сколько указано в операнде
5. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление
6. В отладке можно видеть текущие(во время останова) значения регистров, их адреса. С помощью `GDB run`
7. `breakpoint` - это преднамеренное прерывание выполнения программы, при котором выполняется вызов отладчика
`checkpoint` - снимок состояния программы
`Watchpoint` - точка останова по данным. Срабатывает, когда меняется значение заданного выражения или переменной

Catchpoint - специальная точка останова, которая срабатывает при достижении специального события, например, C++ исключения или загрузки библиотеки

Call stack (стек вызовов) - хранит информацию об активных процедурах и функциях

8. disassemble - Посмотреть дизассемблированный код программы

run - запуск в режиме отладки

breakpoint - установить точку останова

kill (сокращённо k) прекращает отладку программы

next - переход к следующей точке останова