

Отчет по лабораторной работе №9

Дисциплина: Архитектура компьютера

Федорова Анжелика Игоревна

Содержание

1	Цель работы	6
2	Задание	7
3	Теоретическое введение	8
4	Выполнение лабораторной работы	11
5	Самостоятельная работа	22
6	Выводы	31
7	Список литературы	32

Список иллюстраций

4.1	Создание каталога и файла lab09-1.asm	11
4.2	Копирование файла in_out.asm в нужный каталог	11
4.3	Ввод кода из листинга в lab09-1.asm	12
4.4	Запуск файла lab09-1	12
4.5	Редактирование программы в файле lab09-1	13
4.6	Запуск отредактированного файла lab09-1	13
4.7	Создание файла lab09-2.asm	14
4.8	Получение исполняемого файла lab09-2.asm	14
4.9	Загрузка файла lab09-2.asm в отладчик	15
4.10	Запуск команды run	15
4.11	Установка метки break на инструкцию _start	16
4.12	Просмотр дисассемблированного кода и ввод set disassembly-flavor intel	16
4.13	Включение режима псевдографики	17
4.14	Установка точки останова и проверка с помощью info breakpoints	17
4.15	Выполнение 5-ти инструкций с помощью команды stepi	18
4.16	Замена первых символов в msg1 и msg2	18
4.17	Вывод значений регистра edx в 3-х форматах	19
4.18	Изменения значения регистра ebx с помощью set	19
4.19	Изменения значения регистра ebx с помощью set	19
4.20	Создание lab09-3.asm с содержанием из lab8-2.asm	20
4.21	Загрузка исполняемого файла в gdb с ключом -args	20
4.22	Установка точки останова	20
4.23	Просмотр значений по адресам	21
5.1	Создание файла lab09-4.asm	22
5.2	Загрузка исполняемого файла в gdb с ключом -args	23
5.3	Запуск отредактированной программы	23
5.4	Создание файла	25
5.5	Ввод кода программы	25
5.6	Создание исполняемого файла lab09-5	26
5.7	Запуск программы в отладчике	26
5.8	Установка break	26
5.9	Установка режима псевдографики	27
5.10	Изменение значения регистра ebx после суммы с регистром eax	27
5.11	Изменение значения регистров после операции умножения	28
5.12	Исправление кода в программе	28

5.13	Запуск исправленного файла lab09-5	29
------	----------------------------------------------	----

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

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. 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.

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки.

Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

Команда `run` (сокращённо `r`) — запускает отлаживаемую программу в оболочке GDB.

Команда `kill` (сокращённо `k`) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. Если в ответ введено `y` (то есть «да»), отладка программы прекращается. Командой `run` её можно начать заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точки

отлова (catchpoints) сохраняются.

Для выхода из отладчика используется команда `quit` (или сокращённо `q`).

Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом `-g`.

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка».

Информацию о всех установленных точках останова можно вывести командой `info` (кратко `i`).

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой `disable`.

Обратно точка останова активируется командой `enable`.

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды `delete`.

Для продолжения остановленной программы используется команда `continue` (`c`). Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число `N`, которое указывает отладчику проигнорировать `N – 1` точку останова (выполнение остановится на `N`-й точке).

Команда `stepi` (кратко `sl`) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию.

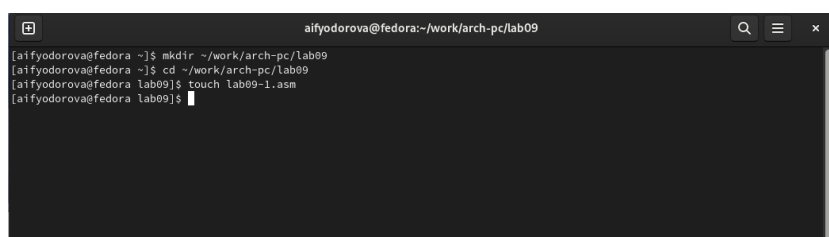
Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом. Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При

этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `еір` адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы. Подпрограмма завершается инструкцией `ret`, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией `call`, и заносит его в `еір`. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией `call`.

4 Выполнение лабораторной работы

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm.(рис. fig:001).



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

Рис. 4.1: Создание каталога и файла lab09-1.asm

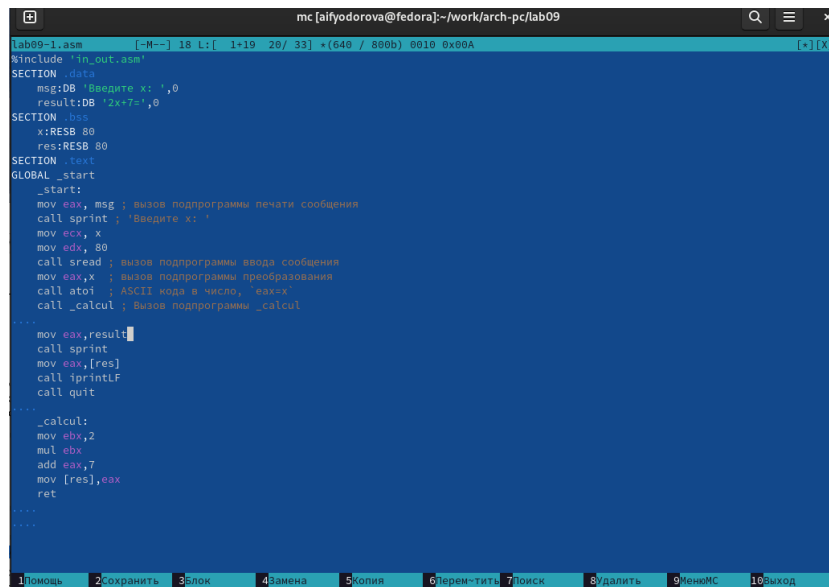
Копирую файл in_out.asm в каталог ~/work/arch-pc/lab09 для работы.(рис. fig:002)



```
[aifyodorova@fedora lab09]$ cp in_out.asm ~/work/arch-pc/lab09
[aifyodorova@fedora lab09]$
```

Рис. 4.2: Копирование файла in_out.asm в нужный каталог

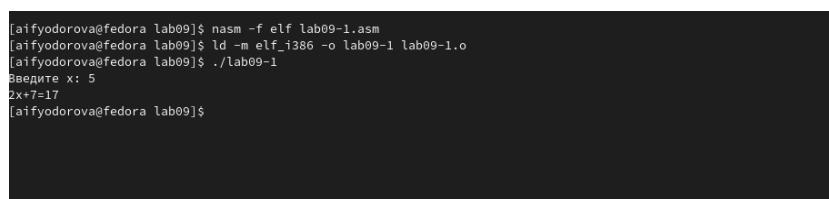
Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1(рис. fig:003)



```
lab09-1.asm [~M--] 18 L: [ 1+19 20/ 33] +(640 / 800b) 0010 0x00A [~] [X]
#include "in_out.asm"
SECTION .data
    msg:DB 'Введите x: ',0
    result:DB '2x+7a',0
SECTION .bss
    x:RESB 80
    res:RESB 80
SECTION .text
GLOBAL _start
_start:
    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 iprintf
    call quit
    ...
    _calcul:
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [res], eax
    ret
    ...
    ...
1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перем-тить 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 4.3: Ввод кода из листинга в lab09-1.asm

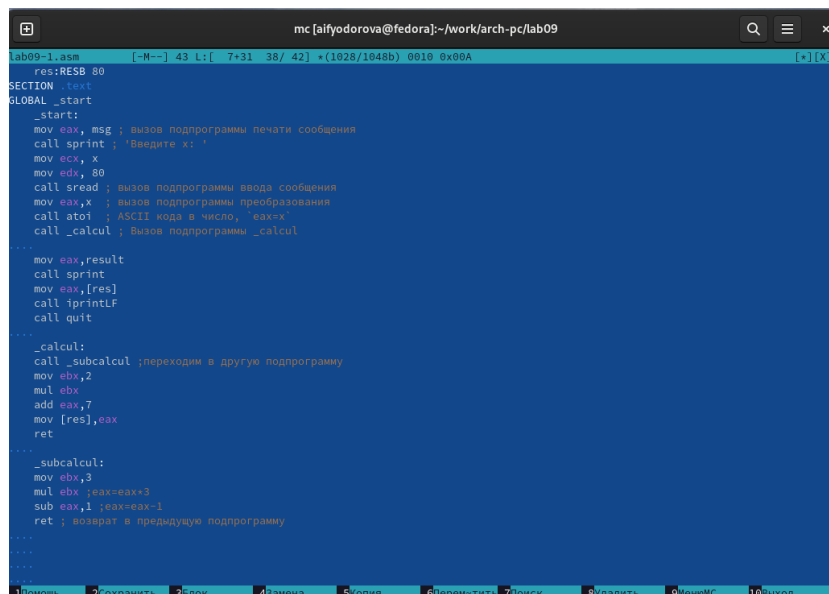
Создаю исполняемый файл и проверяю его работу. (рис. fig:004)



```
[aifedorova@fedora lab09]$ nasm -f elf lab09-1.asm
[aifedorova@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[aifedorova@fedora lab09]$ ./lab09-1
Введите x: 5
2x+7=17
[aifedorova@fedora lab09]$
```

Рис. 4.4: Запуск файла lab09-1

Программа выводит решение уравнения с подстановкой введенного аргумента в переменную x. Теперь я 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. fig:005)



```
lab09-1.asm [-M--] 43 L: [ 7+31 38/ 42] * (1028/1048b) 0010 0x00A [+][X]
res:RESB 80
SECTION .text
GLOBAL _start
_start:
    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 !printLF
    call quit
    ...
    _calcul:
    call _subcalcul ;переходим в другую подпрограмму
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [res], eax
    ret
    ...
    _subcalcul:
    mov ebx, 3
    mul ebx ;eax=eax*3
    sub eax, 1 ;eax=eax-1
    ret ; возврат в предыдущую подпрограмму
    ...
    ...
    ...
1 Помощь 2 Сохранить 3 Блок 4 Замена 5 Копия 6 Переименовать 7 Поиск 8 Удалить 9 Меню MS 10 Выход
```

Рис. 4.5: Редактирование программы в файле lab09-1

Создаю исполняемый файл и проверяю его работу. (рис. fig:006)

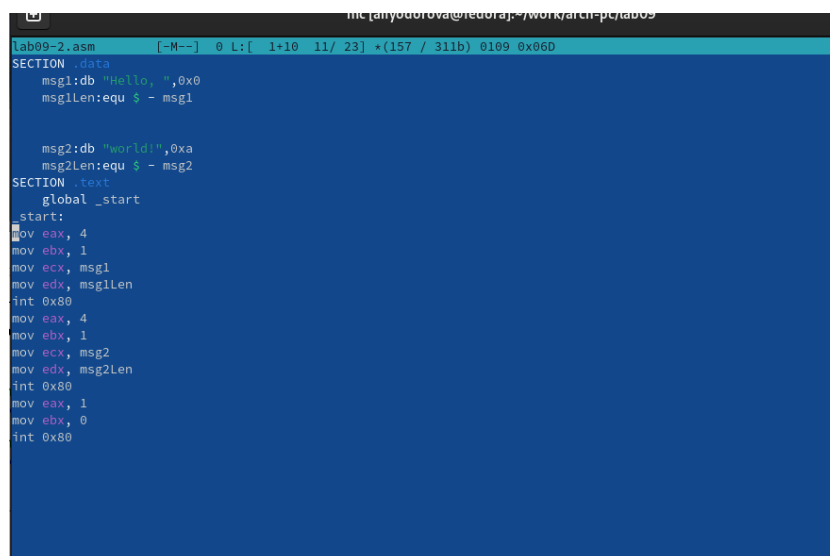


```
aifyodorova@fedora:~/work/arch-pc/lab09
[aifyodorova@fedora lab09]$ nasm -f elf lab09-1.asm
[aifyodorova@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[aifyodorova@fedora lab09]$ ./lab09-1
Введите x: 3
2(3x-1)+7=23
[aifyodorova@fedora lab09]$
```

Рис. 4.6: Запуск отредактированного файла lab09-1

Программа успешно выводит верный ответ функции $f(x) = 2(3x - 1) + 7$

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2.(рис. fig:007)

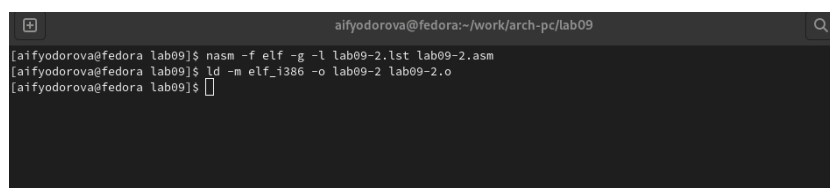


```
lab09-2.asm [-M--] 0 L: [ 1+10 11/ 23] *(157 / 311b) 0109 0x060
SECTION .data
    msg1:db "Hello, ",0x0
    msg1len:equ $ - msg1

    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. 4.7: Создание файла lab09-2.asm

Получаю исполняемый файл для работы с GDB с ключом '-g'.(рис. fig:008)



```
aifyodorova@fedora:~/work/arch-pc/lab09
[aifyodorova@fedora lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[aifyodorova@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
[aifyodorova@fedora lab09]$
```

Рис. 4.8: Получение исполняемого файла lab09-2.asm

Загружаю исполняемый файл в отладчик gdb.(рис. fig:009)

```
aifyodorova@fedora:~/work/arch-pc/lab09 — gdb lab09-2
[aiifyodorova@fedora lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[aiifyodorova@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
[aiifyodorova@fedora lab09]$ gdb lab09-2
GNU gdb (GDB) Fedora Linux 13.1-2.fc38
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) run
Starting program: /home/aifyodorova/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.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 78125) exited normally]
(gdb)
```

Рис. 4.9: Загрузка файла lab09-2.asm в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run.(рис. fig:010)

```
aifyodorova@fedora:~/work/arch-pc/lab09 — gdb lab09-2
[aiifyodorova@fedora lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[aiifyodorova@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
[aiifyodorova@fedora lab09]$ gdb lab09-2
GNU gdb (GDB) Fedora Linux 13.1-2.fc38
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) run
Starting program: /home/aifyodorova/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.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 78125) exited normally]
(gdb)
```

Рис. 4.10: Запуск команды run

Для более подробного анализа программы устанавливаю брейкпоинт на метку _start и запускаю её. (рис. fig:011)

```
[Inferior 1 (process 78125) exited normally]
(gdb) break _start
Breakpoint 1 at 0x04010e9: file lab09-2.asm, line 11.
(gdb) run
Starting program: /home/sifedorova/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:11
11      mov ebx, 4
```

Рис. 4.11: Установка метки break на инструкцию _start

Просматриваю дисассимилированный код программы с помощью команды disassemble, начиная с метки _start, и переключаюсь на отображение команд с синтаксисом Intel, введя команду set disassembly-flavor intel (рис. fig:012)

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x04010e9 <+0>: mov    $0x4,%eax
0x04010e9 <+5>: mov    $0x1,%ebx
0x04010ea <+10>: mov    $0x402118,%ecx
0x04010ef <+15>: mov    $0x8,%edx
0x04010f4 <+20>: int    $0x80
0x04010f6 <+22>: mov    $0x4,%eax
0x04010f6 <+27>: mov    $0x1,%ebx
0x0401100 <+32>: mov    $0x402118,%ecx
0x0401105 <+37>: mov    $0x7,%edx
0x040110a <+42>: int    $0x80
0x040110c <+44>: mov    $0x1,%eax
0x0401111 <+49>: mov    $0x8,%ebx
0x0401116 <+54>: int    $0x80
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x04010e9 <+0>: mov    eax,0x4
0x04010e9 <+5>: mov    ebx,0x1
0x04010ea <+10>: mov    ecx,0x402118
0x04010ef <+15>: mov    edx,0x8
0x04010f4 <+20>: int    0x80
0x04010f6 <+22>: mov    eax,0x4
0x04010f6 <+27>: mov    ebx,0x1
0x0401100 <+32>: mov    ecx,0x402118
0x0401105 <+37>: mov    edx,0x7
0x040110a <+42>: int    0x80
0x040110c <+44>: mov    eax,0x1
0x0401111 <+49>: mov    ebx,0x8
0x0401116 <+54>: int    0x80
End of assembler dump.
```

Рис. 4.12: Просмотр дисассимилированного кода и ввод set disassembly-flavor intel

В режиме АТТ имена регистров начинаются с символа %, а имена операндов с \$, в то время как в Intel используется привычный нам синтаксис.

Включаю режим псевдографики для более удобного анализа программы с помощью команд layout asm и layout regs.(рис. fig:013)

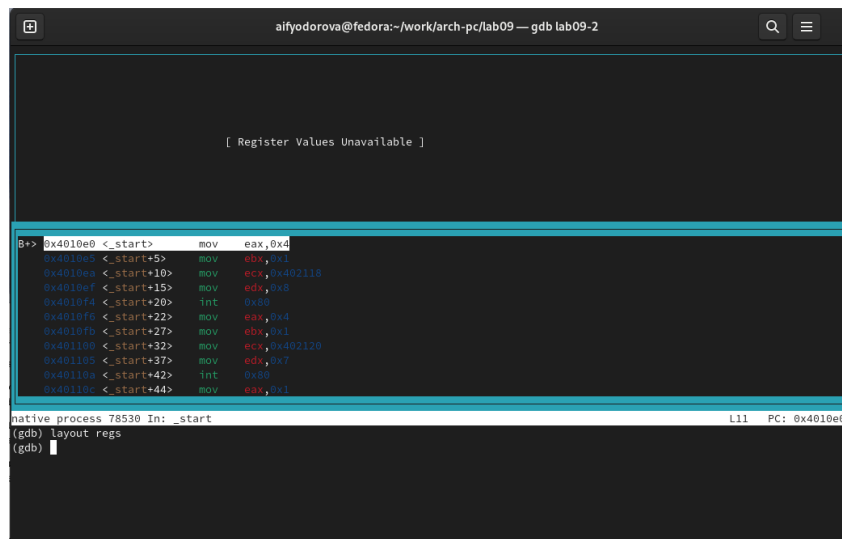


Рис. 4.13: Включение режима псевдографики

Проверяю, что точка останова по имени метки `_start` установлена с помощью команды `info breakpoints` и устанавливаю еще одну точку останова по адресу инструкции `mov ebx,0x0`. Просматриваю информацию о всех установленных точках останова.(рис. fig:014)

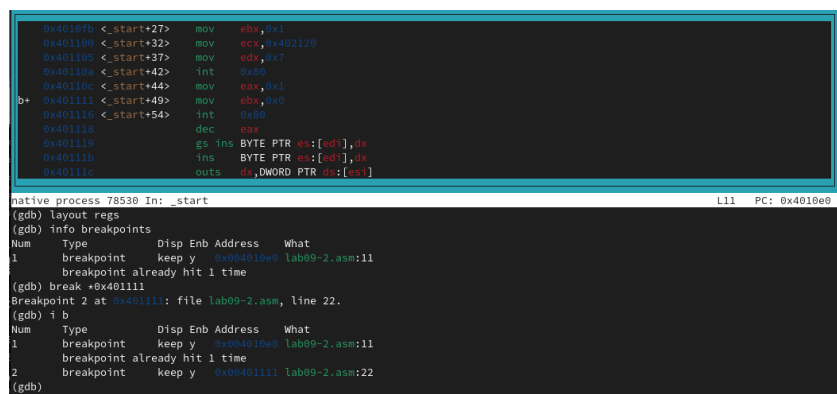


Рис. 4.14: Установка точки остановки и проверка с помощью `info breakpoints`

Выполняю 5 инструкций с помощью команды `stepi` и слежу за изменением значений регистров.(рис. fig:015)

```

aifyodorova@fedora:~/work/arch-pc/lab09 — gdb lab09-2
eax 0x4 4 ecx 0x402118 4202776
eax 0x1 1 ecx 0x402120 4202784
edx 0x7 7 ebp 0x0 0
esi 0x0 0 edi 0x0 0
eip 0x4010fb 0x4010fb <_start+27> eflags 0x202 [ IF ]
cs 0x23 35 l1 49 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

B> 0x4010eb <_start> mov eax,0x4
0x4010fb <_start+22> mov ebx,0x1
0x4010fb <_start+27> mov ecx,0x402118
0x401100 <_start+32> mov edx,0x8 20
0x401105 <_start+37> int 0x80 7
0x40110a <_start+42> mov eax,0x4
0x40110e <_start+44> mov ebx,0x1
B> 0x401111 <_start+49> mov ebx,0x402120
0x401116 <_start+54> int 0x80
0x401118 dec ecx
0x401119 gs ins BYTE PTR es:[edi],dx
0x40111a inc BYTE PTR esi:[edi],dx

native process 78530 In: _start L17 PC: 0x402118
breakpoint already hit 1 time
Num Type Disp Enb Address What
1 breakpoint keep y 0x084010eb lab09-2.asm:11
breakpoint already hit 1 time
2 breakpoint keep y 0x08401111 lab09-2.asm:22
(gdb) si 1
(gdb) si 2
(gdb) si 3
(gdb) si 4
world!
(gdb) si 5
Breakpoint 2, _start () at lab09-2.asm:22
(gdb)

```

Рис. 4.15: Выполнение 5-ти инструкций с помощью команды stepi

Значения регистров изменяются в соответствии с кодом программы. В зависимости от того на какую строку кода я попаду с помощью команды stepi(сколько шагов поставлю от точки останова) от того будет зависеть значение регистра. С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2.(рис. fig:016)

```

(gdb) set (char)&msg1='h'
(gdb) x/1sb &msg1
0x402118 <msg1>: "hello, "
(gdb) set (char)&msg2='r'
(gdb) x/1sb &msg2
0x402119 <msg2>: "world!\n\034"
(gdb)

```

Рис. 4.16: Замена первых символов в msg1 и msg2

Вывожу в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра edx с помощью команды print p/F \$val.(рис. fig:017)

```
(gdb) p/s $edx
$1 = 7
(gdb) p/t $edx
$2 = 111
(gdb) p/x $edx
$3 = 0x7
(gdb)
```

Рис. 4.17: Вывод значений регистра edx в 3-х форматах

С помощью команды set изменяю значение регистра ebx в соответствии с заданием. (рис. fig:018)

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

Рис. 4.18: Изменения значения регистра ebx с помощью set

Разница вывода команд p/s \$ebx отличается тем, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется. Завершаю выполнение программы с помощью команды continue и выхожу из GDB с помощью команды quit (рис. fig:019)

The screenshot shows the GDB interface with the following content:

eax	0x4	4	ecx	0x402118	4202776
eax	0x1	1	ecx	0x402120	4202784
edx	0x7	7	ebx	0x2	2
esi	0x0	0	edi	0x0	0
ebp	0x4010fb	0x4010fb <_start+27>	eflags	0x202	[IF]
ebp	0x401111	0x401111 <_start+49>	ss	0x2b	43
ds	0x2b	43	es	0x2b	43
fs	0x0	0	gs	0x0	0

```

B+ 0x4010e0 <_start> mov    eax,0x4
B+ 0x401111 <_start+49> mov    ebx,0x0
   0x401116 <_start+54> int    0x0  02115

0x4
02120

native process 78530 In: _start
No process In: 1 time
$2 = 111
(gdb) p/x $edx
$3 = 0x7
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb) continue
Continuing.
[Inferior 1 (process 78530) exited normally]
(gdb)

```

Рис. 4.19: Изменения значения регистра ebx с помощью set

Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09-3.asm и создаю исполняемый файл. (рис. fig:020)

```
[aifyodorova@fedora lab09]$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
[aifyodorova@fedora lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[aifyodorova@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Рис. 4.20: Создание lab09-3.asm с содержанием из lab8-2.asm

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа `-args` (рис. fig:021)

```
[aifyodorova@fedora lab09]$ gdb --args lab09-3 1 2 '3'
GNU gdb (GDB) Fedora Linux 13.1-2.fc38
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/>.
```

Рис. 4.21: Загрузка исполняемого файла в gdb с ключом `-args`

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее.(рис. fig:022)

```
(gdb) b _start
Breakpoint 1 at 0x4011a8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/aifyodorova/work/arch-pc/lab09/lab09-3 1 2 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 lab09-3.asm:5
5      pop     ecx ; Извлекаем из стека в 'ecx' количество
```

Рис. 4.22: Установка точки останова

Посматриваю вершину стека и позиции стека по их адресам. (рис. fig:023)


```
(gdb) x/x $esp
0x00000004
(gdb) x/s *(void**)($esp + 4)
0x00000000: "/home/aifedorova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0x00000000: "1"
(gdb) x/s *(void**)($esp + 12)
0x00000000: "2"
(gdb) x/s *(void**)($esp + 16)
0x00000000: "3"
(gdb) x/s *(void**)($esp + 20)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.23: Просмотр значений по адресам

Шаг изменения адреса равен 4, т.к количество аргументов командной строки равно 4.

5 Самостоятельная работа

1. Создаю файл lab09-4.asm для самостоятельной работы, скопировав в него содержание файла lab08-4.asm из предыдущей лабораторной работы. (рис. fig:024)



```
aifyodorova@fedora:~/work/arch-pc/lab07
[aifyodorova@fedora lab07]$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
[aifyodorova@fedora lab07]$
```

Рис. 5.1: Создание файла lab09-4.asm

Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму.(рис. fig:025)

```

lab09-4.asm [0.4s] [46 Lc] [1+20 21/ 30] [1026/10000] 0010 0x00A
#include "in_out.asm"
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
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
; промежуточной суммы
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку _end)
pop eax ; извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
call _calculus ; вызываем подпрограмму _calculus
loop next ; переход к обработке следующего аргумента

_end:
mov eax , msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр eax
call sprintf ; печатаем результат
call quit ; завершение программы

_calculus: ; исполнение подпрограммы, которая работает с аргументами
mov ebx,15
mul ebx
sub eax,9
add esi, eax
ret ; возвращение в основную программу

```

Рис. 5.2: Загрузка исполняемого файла в gdb с ключом `–args`

Запускаю код и проверяю, что она работает корректно. (рис. fig:026)

```

[aifyodorova@fedora lab09]$ ./lab09-4 3 4 5
Результат: 153

```

Рис. 5.3: Запуск отредактированной программы

Код программы:

```

#include 'in_out.asm'

SECTION .data
msg db "Результат: ",0

SECTION .text
global _start
_start:

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 _calculus ;вызываю подпрограмму _calculus
loop next ; переход к обработке следующего аргумента

_end:
mov eax , msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр eax
call iprintLF ; печать результата
call quit ; завершение программы

_calculus: ; исполнение подпрограммы, которая работает с аргументами

mov ebx,15
mul ebx
sub eax,9
add esi, eax

```


ret ; возвращение в основную программу

2. Создаю файл lab09-5.asm для второго самостоятельного задания (рис. fig:027)

```
[aifyodorova@fedora lab09]$ touch lab09-5.asm
[aifyodorova@fedora lab09]$
```

Рис. 5.4: Создание файла

Ввожу в файл lab09-5.asm текст программы из листинга 9.3. (рис. fig:028)

```
lab09-5.asm [-M--] 10 L:[ 1+10 11/ 23] *(184 / 363b) 0010 0x00A
#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. 5.5: Ввод кода программы

При корректной работе программы должно выводиться “25”. Создаю исполняемый файл с ключом -g и запускаю его. (рис. fig:029)

```
[aifyodorova@fedora lab09]$ nasm -f elf -g -l lab09-5.lst lab09-5.asm
[aifyodorova@fedora lab09]$ ld -m elf_i386 -o lab09-5 lab09-5.o
[aifyodorova@fedora lab09]$ gdb lab09-5
```

Рис. 5.6: Создание исполняемого файла lab09-5

Запускаю данную программу с помощью команды run и вижу результат работы кода в файле (рис. fig:030)

```
(gdb) run
Starting program: /home/aifyodorova/work/arch-pc/lab09/lab09-5

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.
Результат: 10
```

Рис. 5.7: Запуск программы в отладчике

Ставлю точку остановки на строку с инструкцией _start (рис. fig:031)

```
[Inferior 1 (process 85326) exited normally]
(gdb) break _start
```

Рис. 5.8: Установка break

Устанавливаю также режим псевдографики с помощью команд layout asm и layout regs (рис. fig:032)

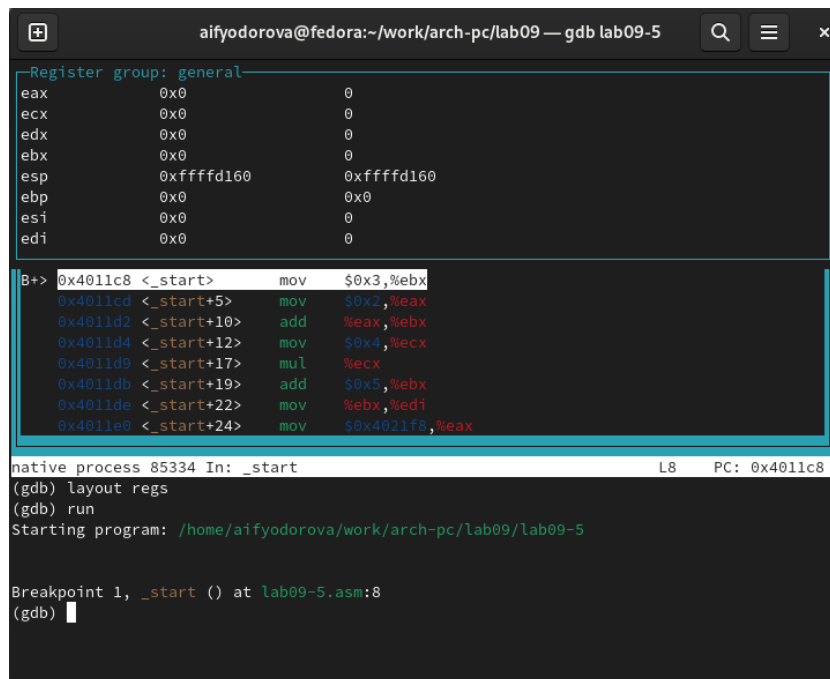


Рис. 5.9: Установка режима псевдографики

Далее я запускаю программу и с помощью команды `stepi` внимательно изучаю изменения в регистрах во время этапов их суммы (рис. fig:033) и умножения на `ecx` (рис. fig:034).

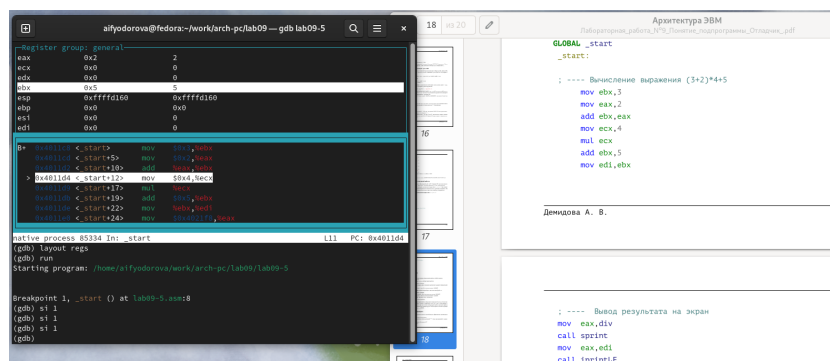


Рис. 5.10: Изменение значения регистра `ebx` после суммы с регистром `eax`

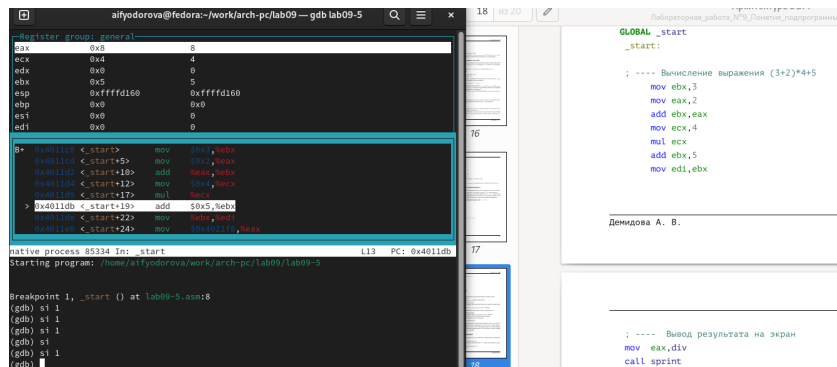


Рис. 5.11: Изменение значения регистров после операции умножения

Я вижу, что регистр ebx принял значение 5 после суммы с eax, однако после строки с умножением на ecx именно значение, записанное в регистре eax увеличилось в 4 раза. Поэтому после увеличения ebx на 5 конечный ответ получается 10. Захожу в файл lab09-5.asm и исправляю код программы, меняя местами в строке сложения ebx и eax эти регистры местами. (рис. fig:035)

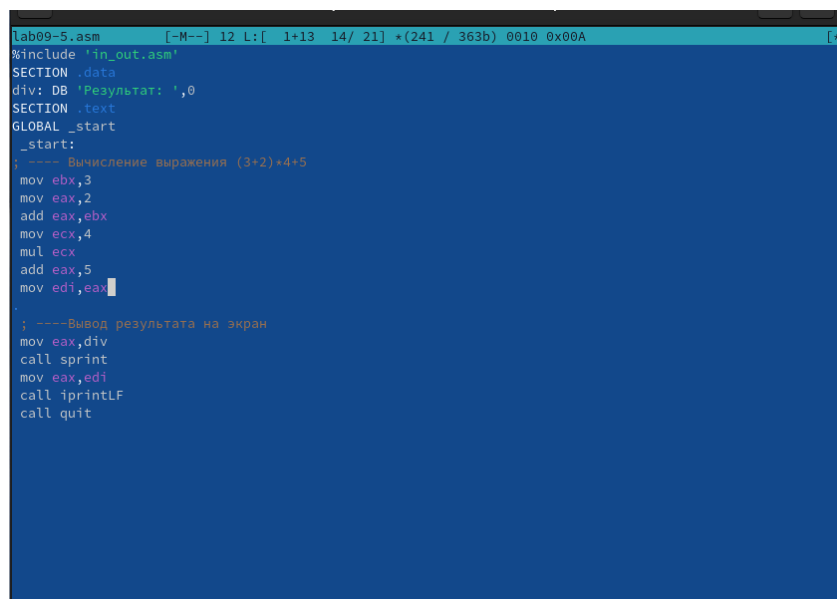


Рис. 5.12: Исправление кода в программе

Теперь создаю снова исполняемый файл и запускаю его. (рис. fig:036)

```
[aifyodorova@fedora lab09]$ nasm -f elf lab09-5.asm
[aifyodorova@fedora lab09]$ ld -m elf_i386 -o lab09-5 lab09-5.o
[aifyodorova@fedora lab09]$ ./lab09-5
Результат: 25
[aifyodorova@fedora lab09]$
```

Рис. 5.13: Запуск исправленного файла lab09-5

Вижу, что теперь программа выводит правильный ответ.

Код 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
```

6 Выводы

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

7 Список литературы

Лабораторная работа №9. Понятие подпрограммы.Отладчик GDB.