

Отчёт по лабораторной работе

Дисциплина: Архитектура компьютера

Вакутайпа Милдред

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выполнение самостоятельной работы	19
5	Выводы	28
6	Список литературы	29

Список иллюстраций

3.1	Рис 1	7
3.2	Рис 2	8
3.3	Рис 3	8
3.4	Рис 4	9
3.5	Рис 5	10
3.6	Рис 6	10
3.7	Рис 7	10
3.8	Рис 8	11
3.9	Рис 9	11
3.10	Рис 10	11
3.11	Рис 11	12
3.12	Рис 12	12
3.13	Рис 13	13
3.14	Рис 14	13
3.15	Рис 15	14
3.16	Рис 16	14
3.17	Рис 17	15
3.18	Рис 18	15
3.19	Рис 19	15
3.20	Рис 20	15
3.21	Рис 21	16
3.22	Рис 22	16
3.23	Рис 23	16
3.24	Рис 24	16
3.25	Рис 25	17
3.26	Рис 26	17
3.27	Рис 27	17
3.28	Рис 28	18
4.1	Рис 29	19
4.2	Рис 30	20
4.3	Рис 31	20
4.4	Рис 32	23
4.5	Рис 33	23

4.6	Рис 34	23
4.7	Рис 35	24
4.8	Рис 36	24
4.9	Рис 37	25
4.10	Рис 38	25
4.11	Рис 39	26

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 программ с помощью GDB

3 Выполнение лабораторной работы

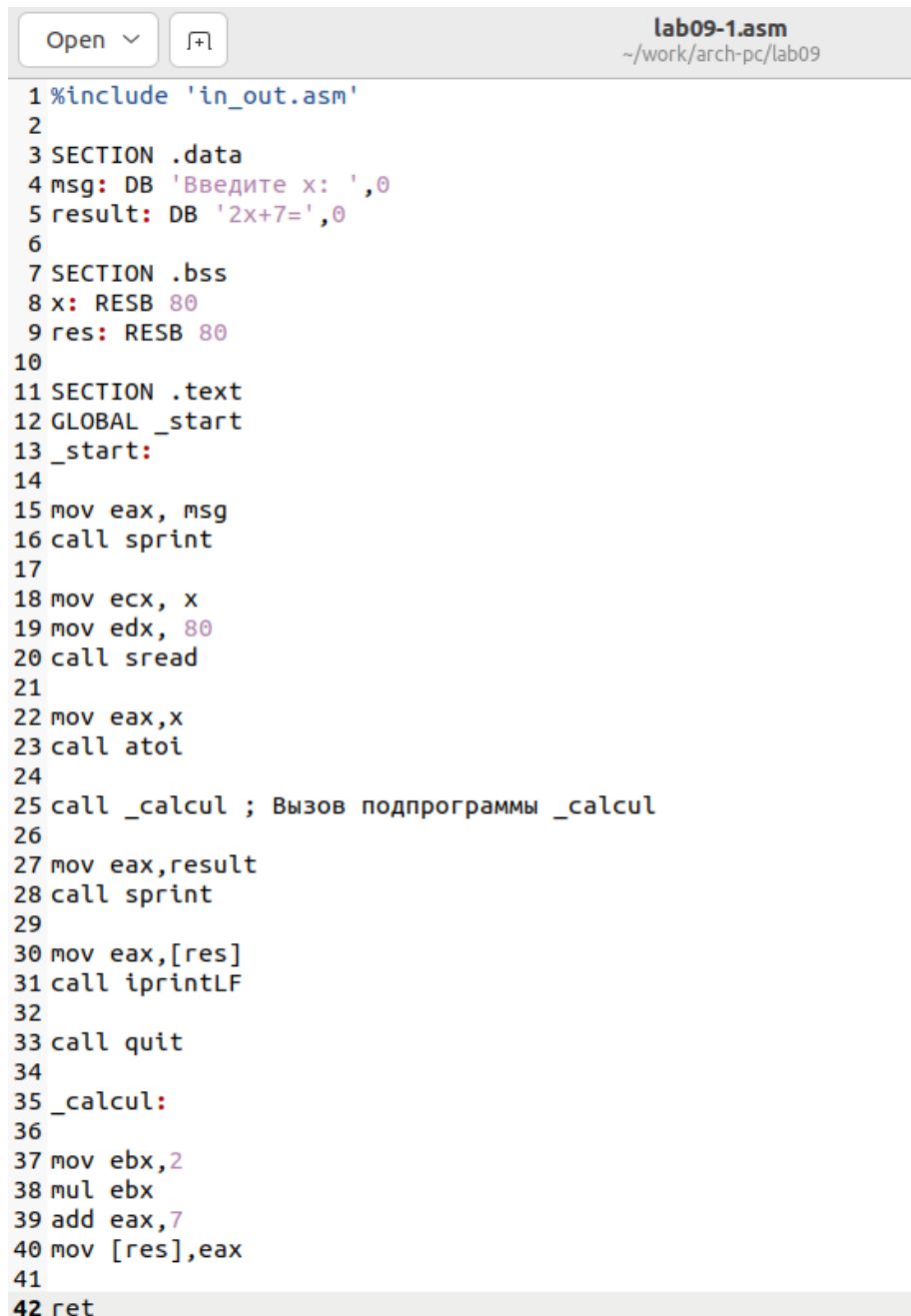
Реализация подпрограмм в NASM

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

```
mwakutaipa@mwakutaipa:~$ mkdir ~/work/arch-pc/lab09
mwakutaipa@mwakutaipa:~$ cd ~/work/arch-pc/lab09
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$ touch lab09-1.asm
```

Рис. 3.1: Рис 1

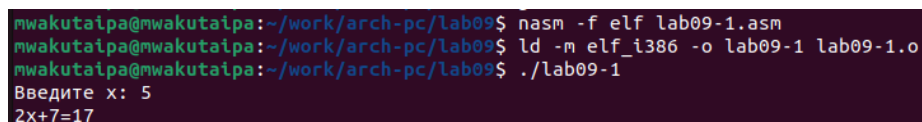
Ввожу в файл lab09-1.asm текст программы для вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы `_calcul`:



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg: DB 'Введите x: ',0
5 result: DB '2x+7=',0
6
7 SECTION .bss
8 x: RESB 80
9 res: RESB 80
10
11 SECTION .text
12 GLOBAL _start
13 _start:
14
15 mov eax, msg
16 call sprint
17
18 mov ecx, x
19 mov edx, 80
20 call sread
21
22 mov eax, x
23 call atoi
24
25 call _calcul ; Вызов подпрограммы _calcul
26
27 mov eax, result
28 call sprint
29
30 mov eax, [res]
31 call iprintLF
32
33 call quit
34
35 _calcul:
36
37 mov ebx, 2
38 mul ebx
39 add eax, 7
40 mov [res], eax
41
42 ret
```

Рис. 3.2: Рис 2

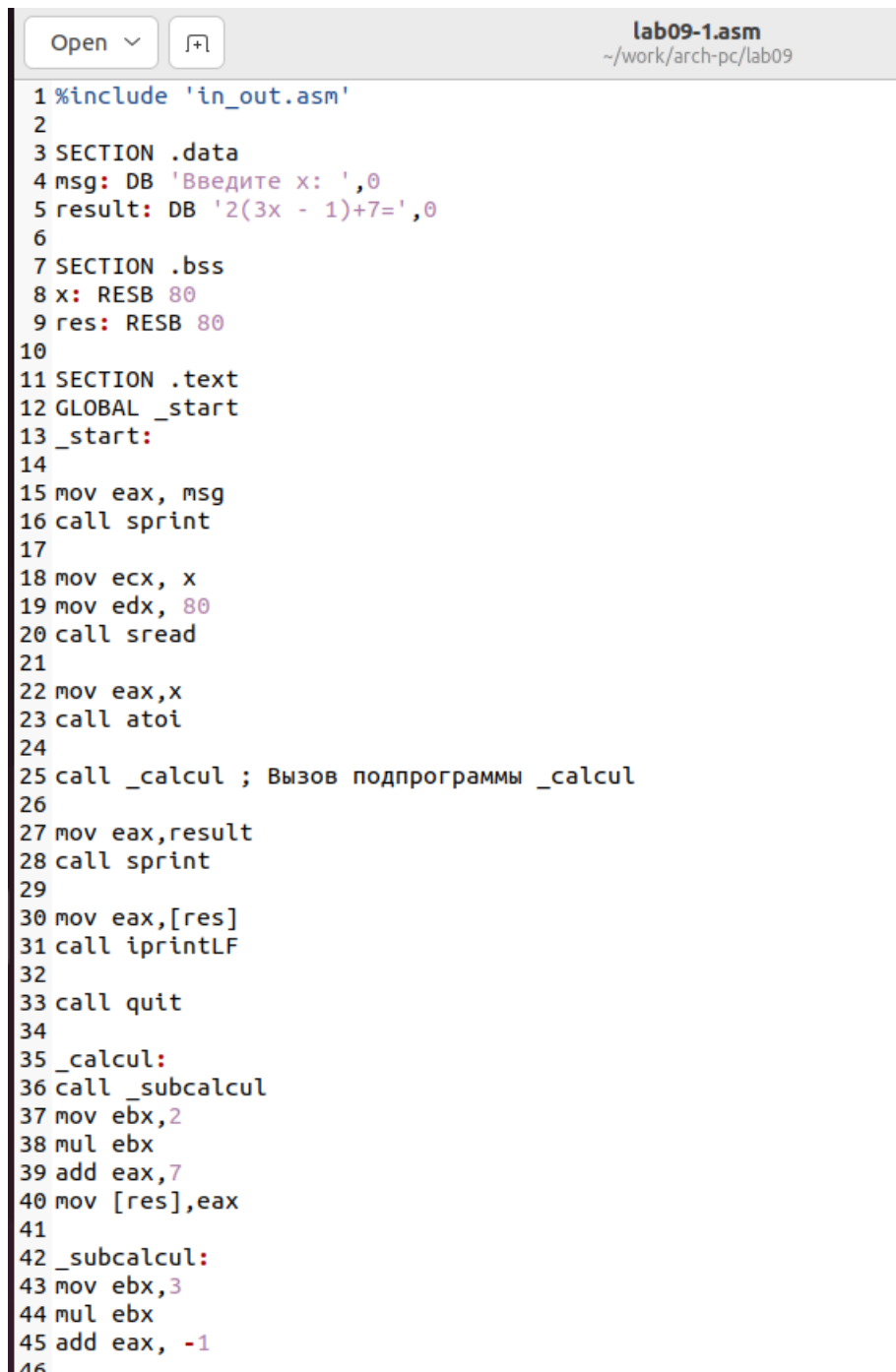
Создаю исполняемый файл и проверяю его работу:



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

Рис. 3.3: Рис 3

Изменяю текст программы, добавляя подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$:



```
lab09-1.asm
~/work/arch-pc/lab09

1 %include 'in_out.asm'
2
3 SECTION .data
4 msg: DB 'Введите x: ',0
5 result: DB '2(3x - 1)+7=',0
6
7 SECTION .bss
8 x: RESB 80
9 res: RESB 80
10
11 SECTION .text
12 GLOBAL _start
13 _start:
14
15 mov eax, msg
16 call sprint
17
18 mov ecx, x
19 mov edx, 80
20 call sread
21
22 mov eax, x
23 call atoi
24
25 call _calcul ; Вызов подпрограммы _calcul
26
27 mov eax, result
28 call sprint
29
30 mov eax, [res]
31 call iprintLF
32
33 call quit
34
35 _calcul:
36 call _subcalcul
37 mov ebx, 2
38 mul ebx
39 add eax, 7
40 mov [res], eax
41
42 _subcalcul:
43 mov ebx, 3
44 mul ebx
45 add eax, -1
46
```

Рис. 3.4: Рис 4

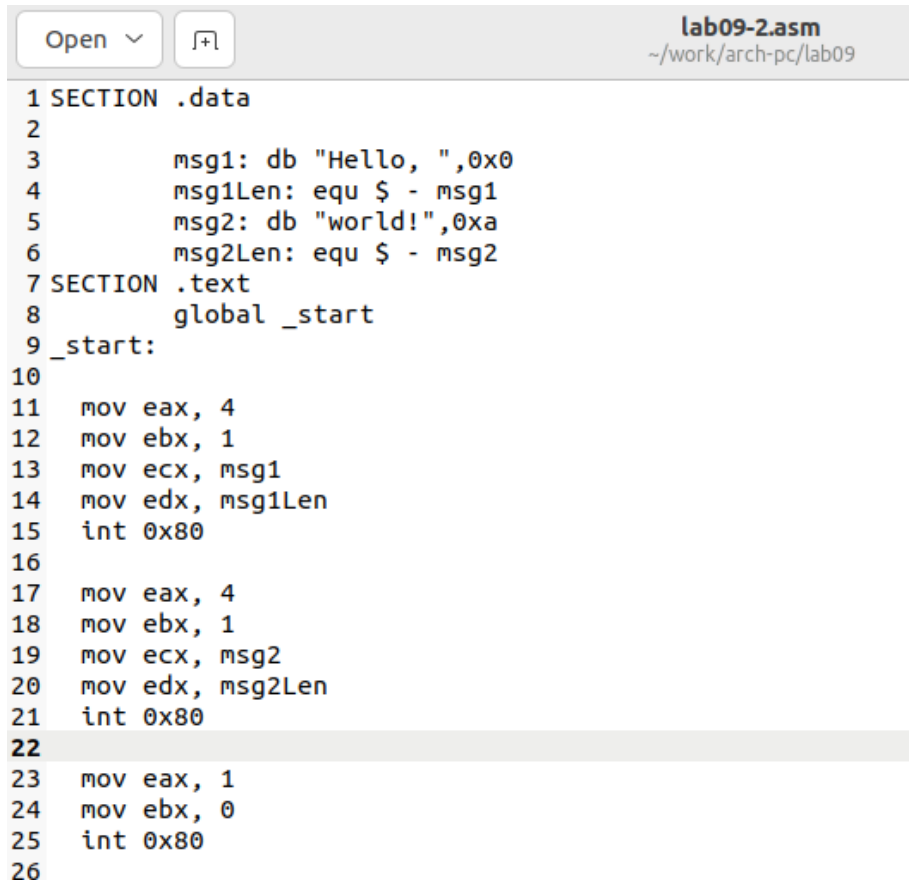
Создаю исполняемый файл и проверяю его работу:

```
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 2
2(3x - 1)+7=17
```

Рис. 3.5: Рис 5

Отладка программ с помощью GDB

Создаю файл lab09-2.asm с текстом программы для печати сообщения Hello world!:



```
lab09-2.asm
~/work/arch-pc/lab09

1 SECTION .data
2
3     msg1: db "Hello, ",0x0
4     msg1Len: equ $ - msg1
5     msg2: db "world!",0xa
6     msg2Len: equ $ - msg2
7 SECTION .text
8     global _start
9 _start:
10
11     mov eax, 4
12     mov ebx, 1
13     mov ecx, msg1
14     mov edx, msg1Len
15     int 0x80
16
17     mov eax, 4
18     mov ebx, 1
19     mov ecx, msg2
20     mov edx, msg2Len
21     int 0x80
22
23     mov eax, 1
24     mov ebx, 0
25     int 0x80
26
```

Рис. 3.6: Рис 6

Создаю исполняемый файл добавляя ключ ‘-g’, для работы с GDB:

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

Рис. 3.7: Рис 7

Загружаю исполняемый файл в отладчик gdb:

```
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
```

Рис. 3.8: Рис 8

Проверяю работу программы, запуская ее в оболочке GDB с помощью команды run:

```
Reading symbols from lab09-2...
(gdb) run
Starting program: /home/mwakutaipa/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 3311) exited normally]
(gdb)
```

Рис. 3.9: Рис 9

После установки брейкпоинт на метку `_start`, запускаю программу для более подробного анализа:

```
[Inferior 1 (process 3311) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 11.
(gdb) run
Starting program: /home/mwakutaipa/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:11
11      mov eax, 4
(gdb) █
```

Рис. 3.10: Рис 10

С помощью команды `disassemble` начиная с метки `_start`, смотрю дисассимилированный код программы:

```
(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. 3.11: Рис 11

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`:

```
(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. 3.12: Рис 12

В синтаксе АТТ первый операнд является источником, а второй является пунктом назначения (напр. `$0x4,%eax`). В синтаксе Intel, первый операнд является пунктом назначения, а второй является источником (напр. `eax,0x4`).

Включаю режим псевдографики для более удобного анализа программы используя `layout asm` и `layout regs`:

```

[ 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 3342 In: _start L11 PC: 0x8049000
(gdb) layout regs

```

Рис. 3.13: Рис 13

В верхней части должно быть названия регистров и их текущие значения, в средней части виден результат дисассимилирования программы и нижняя часть доступна для ввода команд.

Добавление точек останова

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

```

[ 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 3342 In: _start L11 PC: 0x8049000
(gdb) layout regs
(gdb) i b
Num      Type             Disp Enb Address      What
1        breakpoint       keep y 0x08049000 lab09-2.asm:11
breakpoint already hit 1 time
(gdb)

```

Рис. 3.14: Рис 14

Устанавливаю еще одну точку останова по адресу инструкции (`mov ebx,0x0`) используя `break *` и смотрю информацию о всех установленных точках останова:

```

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 3342 In: _start L11 PC: 0x8049000
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 24.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y  0x08049000 lab09-2.asm:11
          breakpoint already hit 1 time
2        breakpoint      keep y  0x08049031 lab09-2.asm:24
(gdb)

```

Рис. 3.15: Рис 15

Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды stepi (si):

```

Register group: general
eax      0x8                                8
ecx      0x804a000                          134520832
edx      0x8                                8
ebx      0x1                                1
esp      0xffffd130                         0xffffd130

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

native process 3342 In: _start L17 PC: 0x8049016
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 24.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y  0x08049000 lab09-2.asm:11
          breakpoint already hit 1 time
2        breakpoint      keep y  0x08049031 lab09-2.asm:24
(gdb) stepi 5
(gdb)

```

Рис. 3.16: Рис 16

С помощью команды x &, смотрю значение переменной msg1 по имени:

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd130 0xffffd130

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 3342 In: _start L17 PC: 0x8049016
ebp      0x0      0x0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--qQuit
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "

```

Рис. 3.17: Рис 17

С помощью команды x/NFU , смотрю значение переменной msg2 по адресу:

```

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

```

Рис. 3.18: Рис 18

Изменяю первый символ переменной msg1 и msg2 с помощью команды set:

```

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

```

Рис. 3.19: Рис 19

```

(gdb) set {char}&msg2='W'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "World!\n\034"
(gdb) 

```

Рис. 3.20: Рис 20

Команда print/F используется для просмотра значений регистров. Вывожу значение регистра edx в двоичном формате, в шестнадцатеричном

формате и в символьном виде:

```
(gdb) p/t $edx
$2 = 1000
(gdb) p/x $edx
$3 = 0x8
(gdb) p/s $edx
$4 = 8
(gdb)
```

Рис. 3.21: Рис 21

С помощью команды set изменяю значение регистра ebx:

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$6 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$7 = 2
(gdb)
```

Рис. 3.22: Рис 22

(gdb) set \$ebx='2' - изменяет значение ebx на значение символа в двоичном формате 2(50). (gdb) set \$ebx=2 изменяет значение ebx на 2.

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:

```
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
```

Рис. 3.23: Рис 23

Создаю исполняемый файл:

```
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$
```

Рис. 3.24: Рис 24

Загружаю исполняемый файл в отладчик с ключом `--args`, указывая аргументы:

```
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумен  
т 2 'аргумент 3'
```

Рис. 3.25: Рис 25

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее:

```
(gdb) b _start  
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 7.  
(gdb) run  
Starting program: /home/mwakutaipa/work/arch-pc/lab09/lab09-3 аргумент1 аргумен  
т 2 аргумент\ 3  
Breakpoint 1, _start () at lab09-3.asm:7
```

Рис. 3.26: Рис 26

Адрес вершины стека храниться в регистре `esp` и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы), здесь число аргументов равно 5:

```
(gdb) x/x $esp  
0xffffd100: 0x00000005  
(gdb)
```

Рис. 3.27: Рис 27

Размер шага изменения адреса по умолчанию равен 4 при отладке программы на архитектуре `x86`. Это связано с тем, что в 32-битных системах адреса памяти обычно представляются как 32-битные числа, и каждый адрес соответствует одному байту:

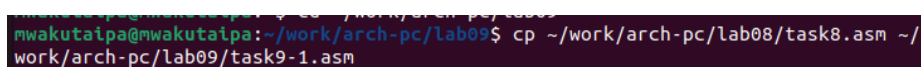
```
(gdb) x/x $esp
0xffffd100:    0x00000005
(gdb) x/s *(void**)($esp +4)
0xffffd2d7:    "/home/mwakutaipa/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd303:    "аргумент1"
(gdb) x/s *(void**)($esp +12)
A syntax error in expression, near `'.
(gdb) x/s *(void**)($esp +12)
0xffffd315:    "аргумент"
(gdb) x/s *(void**)($esp +16)
0xffffd326:    "2"
(gdb) x/s *(void**)($esp +20)
0xffffd328:    "аргумент 3"
(gdb)
```

Рис. 3.28: Рис 28

4 Выполнение самостоятельной работы

Задание 1

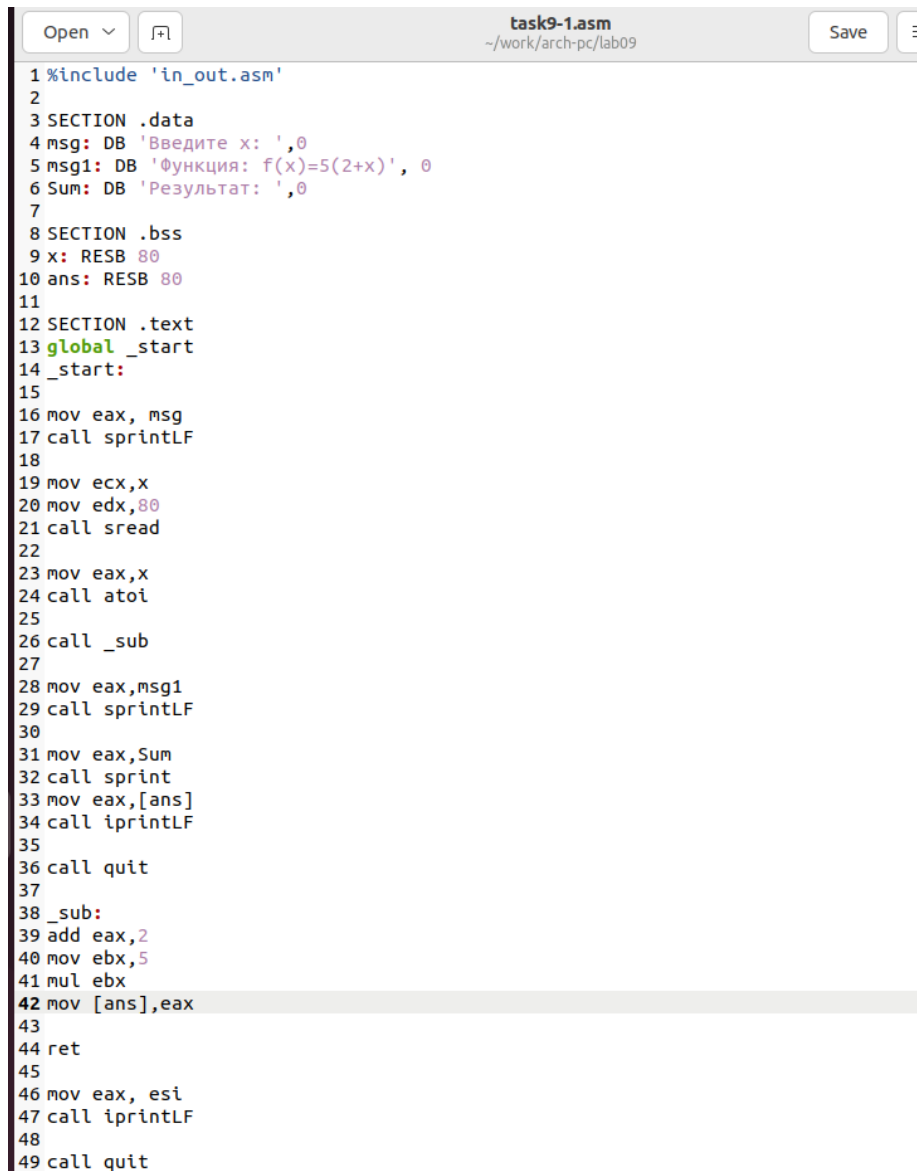
Копирую файл task8.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран сумму значений функции $f(x) = 5(2 + x)$ для некоторых значений x (аргументы командной строки), в файл с именем task9-1.asm:



```
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/task8.asm ~/work/arch-pc/lab09/task9-1.asm
```

Рис. 4.1: Рис 29

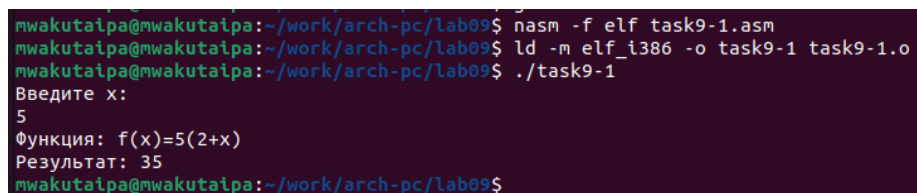
Редактирую программу для вычисления значения функции $f(x)$ как подпрограмму:



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg: DB 'Введите x: ',0
5 msg1: DB 'Функция: f(x)=5(2+x)', 0
6 Sum: DB 'Результат: ',0
7
8 SECTION .bss
9 x: RESB 80
10 ans: RESB 80
11
12 SECTION .text
13 global _start
14 _start:
15
16 mov eax, msg
17 call sprintf
18
19 mov ecx, x
20 mov edx, 80
21 call sread
22
23 mov eax, x
24 call atoi
25
26 call _sub
27
28 mov eax, msg1
29 call sprintf
30
31 mov eax, Sum
32 call sprintf
33 mov eax, [ans]
34 call iprintLF
35
36 call quit
37
38 _sub:
39 add eax, 2
40 mov ebx, 5
41 mul ebx
42 mov [ans], eax
43
44 ret
45
46 mov eax, esi
47 call iprintLF
48
49 call quit
```

Рис. 4.2: Рис 30

Создаю исполняемый файл и проверяю его работу:



```
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$ nasm -f elf task9-1.asm
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$ ld -m elf_i386 -o task9-1 task9-1.o
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$ ./task9-1
Введите x:
5
Функция: f(x)=5(2+x)
Результат: 35
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$
```

Рис. 4.3: Рис 31

Код программы:

```

#include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
msg1: DB 'Функция: f(x)=5(2+x)', 0
Sum: DB 'Результат: ',0

SECTION .bss
x: RESB 80
ans: RESB 80

SECTION .text
global _start
_start:

mov eax, msg
call sprintfLF

mov ecx, x
mov edx, 80
call sread

mov eax, x
call atoi

call _sub

mov eax, msg1

```

```
call sprintLF
```

```
mov eax,Sum
```

```
call sprint
```

```
mov eax,[ans]
```

```
call iprintLF
```

```
call quit
```

```
_sub:
```

```
add eax,2
```

```
mov ebx,5
```

```
mul ebx
```

```
mov [ans],eax
```

```
ret
```

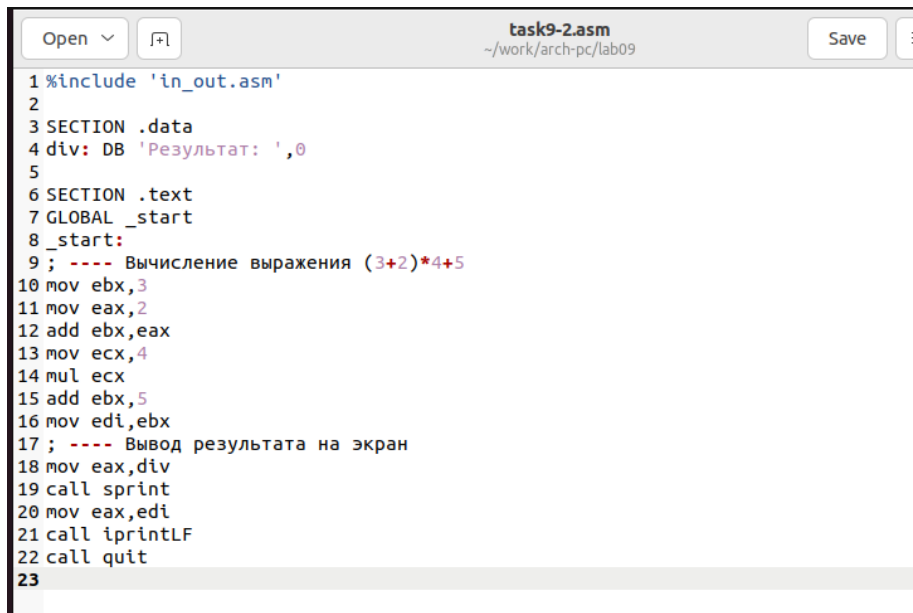
```
mov eax, esi
```

```
call iprintLF
```

```
call quit
```

Задание 2

Создаю файл task9-2.asm и вставляю в него программу вычисления выражения $(3+2) \times 4 + 5$:

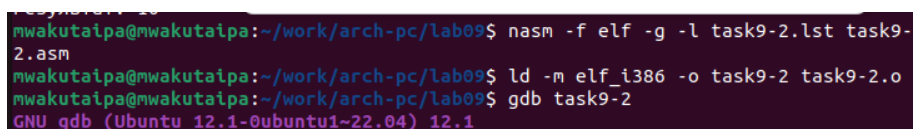


```
task9-2.asm
~/work/arch-pc/lab09
Save

1 %include 'in_out.asm'
2
3 SECTION .data
4 div: DB 'Результат: ',0
5
6 SECTION .text
7 GLOBAL _start
8 _start:
9 ; ---- Вычисление выражения (3+2)*4+5
10 mov ebx,3
11 mov eax,2
12 add ebx,eax
13 mov ecx,4
14 mul ecx
15 add ebx,5
16 mov edi,ebx
17 ; ---- Вывод результата на экран
18 mov eax,div
19 call sprint
20 mov eax,edi
21 call iprintLF
22 call quit
23
```

Рис. 4.4: Рис 32

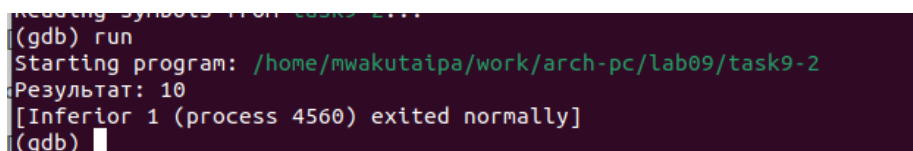
Создаю исполняемый файл:



```
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$ nasm -f elf -g -l task9-2.lst task9-2.asm
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$ ld -m elf_i386 -o task9-2 task9-2.o
mwakutaipa@mwakutaipa:~/work/arch-pc/lab09$ gdb task9-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
```

Рис. 4.5: Рис 33

При запуске она выводит неверный результат:



```
(gdb) run
Starting program: /home/mwakutaipa/work/arch-pc/lab09/task9-2
Результат: 10
[Inferior 1 (process 4560) exited normally]
(gdb)
```

Рис. 4.6: Рис 34

Переключаюсь на отображение команд с Intel'овским синтаксисом:

```

(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 <iprintLF>
   0x08049111 <+41>:   call   0x80490db <quit>

```

Рис. 4.7: Рис 35

Включаю режим псевдографики для более удобного анализа программы используя layout asm:

```

0x80490e7 <quit+12>    ret
b+ 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>
0x804910a <_start+34>   mov     eax,edi
0x804910c <_start+36>   call   0x8049086 <iprintLF>
0x8049111 <_start+41>   call   0x80490db <quit>

```

exec No process in: L?? PC: ??
(gdb)

Рис. 4.8: Рис 36

С помощью layout regs, я могу видеть названия регистров и их текущие значения. Выполняю 6 инструкций с помощью команды stepi (si) и при этом замечаю, что после того как программа суммирует 3 и 2, значение хранится в регистре ebx. Значение регистра ecx(4) умножает на 2(значение регистра eax) а затем суммирует 5 и значение регистра ebx (5):


```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd130 0xffffd130

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>
0x804910a <_start+34> mov    eax,edi

native process 2712 In: _start L16 PC: 0x80490fe
(gdb) layout reg
(gdb) run
Starting program: /home/mwakutaipa/work/arch-pc/lab09/task9-2

Breakpoint 1, _start () at task9-2.asm:10
(gdb) si 6

```

Рис. 4.9: Рис 37

Исправляю код ,чтобы программа выводила правильный ответ:

```

Open  [icon]
1 %include 'in_out.asm'
2
3 SECTION .data
4 div: DB 'Результат: ',0
5
6 SECTION .text
7 GLOBAL _start
8 _start:
9 ; ---- Вычисление выражения (3+2)*4+5
10 mov ebx,3
11 mov eax,2
12 add eax,ebx
13 mov ecx,4
14 mul ecx
15 add eax,5
16 mov edi,eax
17 ; ---- Вывод результата на экран
18 mov eax,div
19 call sprint
20 mov eax,edi
21 call iprintLF
22 call quit
23

```

Рис. 4.10: Рис 38

```
Type "apropos word" to search for commands related to "word"...
Reading symbols from task9-2...
(gdb) run
Starting program: /home/mwakutaipa/work/arch-pc/lab09/task9-2
Результат: 25
[Inferior 1 (process 3400) exited normally]
(gdb)
```

Рис. 4.11: Рис 39

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

5 Выводы

При выполнении данной работы я освоила написание программ с использованием подпрограмм и знакомила методы отладки при помощи GDB и его основные возможности.

6 Список литературы

Архитектура ЭВМ