

Шаблон отчёта по лабораторной работе

архитектура компьютера

мохамед Муса

Цель работы

Цель этой работы - попрактиковаться в языке ассемблера и научиться отлаживать asm-файлы с помощью команды **gdb**.

выполнения лабораторной работы

- Сначала я создал файл lab9-1.asm, скопировал код из pdf и запустил его :

```
bs/lab09/report$ touch lab9-1.asm
liveuser@localhost-live:~/work/study/2023-2024/архитектура компьютера/arch-pc/lab09/report$ gedit lab9-1.asm
liveuser@localhost-live:~/work/study/2023-2024/архитектура компьютера/arch-pc/lab09/report$ nasm -f elf lab9-1.asm
liveuser@localhost-live:~/work/study/2023-2024/архитектура компьютера/arch-pc/lab09/report$ ld -m elf_i386 -o lab9-1 lab9-1.o
liveuser@localhost-live:~/work/study/2023-2024/архитектура компьютера/arch-pc/lab09/report$ ./lab9-1
Введите x: 5
2x+7=17
```

```

lab9-1.asm
~/work/study/2023-2024/архитектура компьютера/arch-pc/labs/lab09/report
Save
x

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

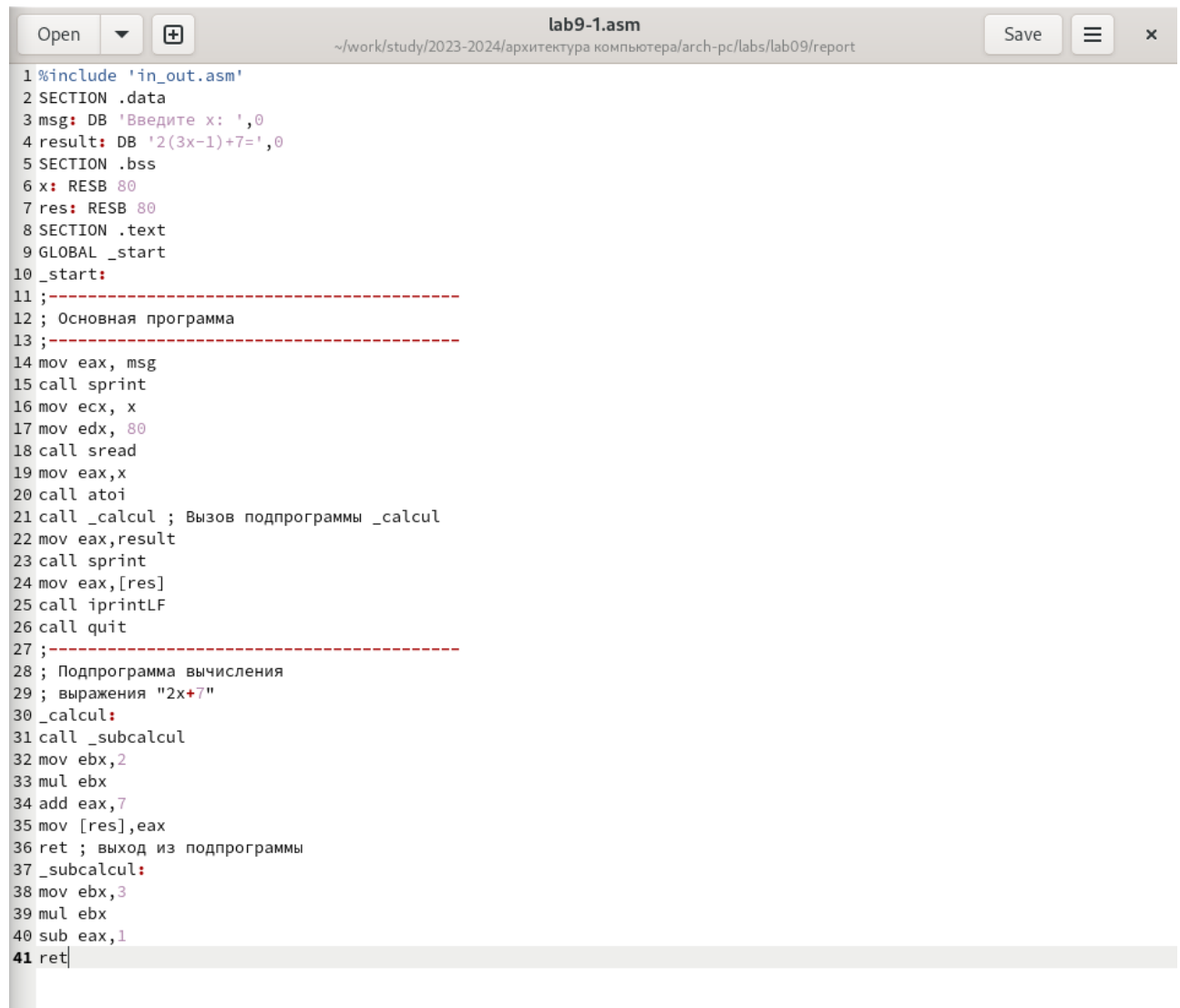
```

- И я внес необходимые изменения из pdf-файл в lab9-1.asm и запустил его снова :

```

bs/lab09/report$ gedit lab9-1.asm
liveuser@localhost-live:~/work/study/2023-2024/архитектура компьютера/arch-pc/la
bs/lab09/report$ nasm -f elf lab9-1.asm
liveuser@localhost-live:~/work/study/2023-2024/архитектура компьютера/arch-pc/la
bs/lab09/report$ ld -m elf_i386 -o lab9-1 lab9-1.o
liveuser@localhost-live:~/work/study/2023-2024/архитектура компьютера/arch-pc/la
bs/lab09/report$ ./lab9-1
Введите x: 7
2(3x-1)+7=47

```



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

- Я создал файл lab9-2.asm и скопировал код из pdf, после чего преобразовал файл из .asm в .asm. Сначала я использовал команду **nasm**, чтобы дать мне возможность использовать команду **gdb** и запустить код в отладчике, а также просмотреть дисассемблированный код с помощью команды **disassemble _start**. И переключился на отображение команд с Intel'овским синтаксисом, введя

команду **set disassembly-flavor intel**:

```

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 52092) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/liveuser/work/study/2023-2024/архитектура компьютера/arch-pc/labs/lab09/report/lab9-2

Breakpoint 1, _start () at lab9-2.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) 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

```

- И я использовал команды **layout asmi** и **layout regs**, чтобы продолжить анализ кода :

```

[ 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

native process 52097 In: _start                                L9      PC: 0x8049000
(gdb) layout regs
(gdb) s

```

- Я установил новую точку останова с помощью команды **break** :

```
(gdb) x/l $b 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)
```

- Используя команду **set**, я изменил значение msg2 :

```
(gdb) set {char}0x804a008='h'
(gdb) x/l $b 0x804a008
0x804a008 <msg2>:      "horld!\n\034"
(gdb)
```

- И я использовал команду **p/""** для редактирования значений в моем коде :

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) p/t $ebx
$2 = 110010
(gdb) p/x $ebx
$3 = 0x32
(gdb)
```

1. Команда **p** в GDB используется для вывода значения выражения. При этом **p/s** указывает формат вывода. Ниже приведены возможные форматы и их значения:

- **p** или **print** — вывести значение выражения в стандартном формате.
- **p/x** — вывести значение выражения в шестнадцатеричном формате.
- **p/d** — вывести значение выражения в десятичном формате.
- **p/o** — вывести значение выражения в восьмеричном формате.
- **p/t** — вывести значение выражения в двоичном формате.
- **p/s** — вывести значение выражения в символьном виде (если возможно).
- Также я создал lab9-3.asm и преобразовал его в файл **.lst**, после чего использовал команду **gdb --args** для отладки файлов с аргументами :

```
liveuser@localhost-live:~/work/study/2023-2024/архитектура компьютера/arch-pc/labs/lab09/report$ gdb --args lab9-3 3 4 '5'
GNU gdb (Fedora Linux) 14.2-1.fc40
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 lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 13.
(gdb) run
Starting program: /home/liveuser/work/study/2023-2024/архитектура компьютера/arch-pc/labs/lab09/report/lab9-3 3 4 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.

Breakpoint 1, _start () at lab9-3.asm:13
13      mov eax,msg1
(gdb) x/x $esp
0xffffcf80: 0x00000004
```

- И я также проверил адрес аргументов :

```
(gdb) x/x $esp
0xffffcf80: 0x00000004
(gdb) x/s *(void**)(esp + 4)
0xffffd145: "/home/liveuser/work/study/2023-2024/архитектура компьютера/arch-pc/labs/lab09/report/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd1b5: "3"
(gdb) x/s *(void**)(esp + 12)
0xffffd1b7: "4"
(gdb) x/s *(void**)(esp + 16)
0xffffd1b9: "5"
(gdb) x/s *(void**)(esp + 20)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Шаг изменения адреса равен 4 (**[esp+4], [esp+8], [esp+12] и т.д.**), потому что стек хранит данные в формате 32-битных значений (4 байта). В системе x86, указатели и данные передаются через стек, который использует 4-байтовые слова для адресации.

Выполнения заданий для самостоятельной работы:

- сначала я написал первую программу и запустил ее :

```
liveuser@localhost-live:~/work/study/2023-2024/архитектура компьютера/arch-pc/labs/lab09/report$ nasm -f elf pr01.asm
liveuser@localhost-live:~/work/study/2023-2024/архитектура компьютера/arch-pc/labs/lab09/report$ ld -m elf_i386 -o pr01 pr01.o
liveuser@localhost-live:~/work/study/2023-2024/архитектура компьютера/arch-pc/labs/lab09/report$ ./pr01
f(x)= 15x + 2
Результат: 0
liveuser@localhost-live:~/work/study/2023-2024/архитектура компьютера/arch-pc/labs/lab09/report$ ./pr01 5
f(x)= 15x + 2
Результат: 77
```

```

Open  [v] [+] *pro1.asm
~/work/study/2023-2024/архитектура компьютера/arch-pc/labs/lab09/report Save [≡] [x]

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)= 15x + 2',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintf
11 pop ecx
12 pop edx
13 sub ecx,1
14 mov esi, 0
15
16 next:
17 cmp ecx,0h
18 jz _end
19 pop eax
20 call atoi
21 call _funk
22 add esi,eax
23
24 loop next
25
26 _end:
27 mov eax, msg
28 call sprint
29 mov eax, esi
30 call iprintLF
31 call quit
32
33 _funk:
34 mov ebx,15
35 mul ebx
36 add eax,2
37 ret

```

- и я написал вторую программу в соответствии с инструкциями, приведенными в pdf-файле и запустил ее:

```

Open  [v] [+] *pro2.asm
~/work/study/2023-2024/архитектура компьютера/arch-pc/labs/lab09/report Save [≡] [x]

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

```

- поскольку в первой программе была допущена ошибка, я использовал команду gdb и исправил ошибку :

Ошибка была в неправильном использовании регистров и операций. Исправленная программа правильно вычисляет выражение, сохраняя промежуточные результаты и корректно использует

регистры для умножения.

Register group: general

eax	0x2	2	ecx	0x0	0
edx	0x0	0	ebx	0x3	3
esp	0xffffcf90	0xffffcf90	ebp	0x0	0x0
esi	0x0	0	edi	0x0	0
eip	0x80490f2	0x80490f2 <_start+10>	eflags	0x10202	[IF RF]
cs	0x23	35	ss	0x2b	43
ds	0x2b	43	es	0x2b	43
fs	0x0	0	gs	0x0	0

B+

0x80490e8 <_start>

mov

\$0x3,%ebx

0x80490ed <_start+5>

mov

\$0x2,%eax

>0x80490f2 <_start+10>

add

%eax,%ebx

0x80490f4 <_start+12>

mov

\$0x4,%ecx

0x80490f9 <_start+17>

mul

%ecx

0x80490fb <_start+19>

add

\$0x5,%ebx

0x80490fe <_start+22>

mov

%ebx,%edi

0x8049100 <_start+24>

mov

\$0x804a000,%eax

0x8049105 <_start+29>

call

0x804900f <sprint>

0x804910a <_start+34>

mov

%edi,%eax

0x804910c <_start+36>

call

0x8049086 <iprintLF>

0x8049111 <_start+41>

call

0x80490db <quit>

0x8049116

add

%al,(%eax)

0x8049118

add

%al,(%eax)

0x804911a

add

%al,(%eax)

native process 53950 In: _start

L10 PC: 0x80490f2

(gdb) layout regs

(gdb) set \$eax =3

Выводы

в этой работе мы узнали, как отлаживать и редактировать asm-файлы с помощью команды gdb