

Министерство образования и науки Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования  
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ  
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)  
Кафедра комплексной информационной безопасности электронно-  
вычислительных систем (БИС)

Программирование на языке Ассемблер

Отчет по лабораторной работе №2  
по дисциплине «Системное программирование»  
1 вариант

Студент гр. 738-1

\_\_\_\_\_ Г.К. Беляев

“ ” \_\_\_\_\_ 2022г.

Принял

Преподаватель кафедры КИБЭВС

Е.О. Калинин

## 1 Введение

Цель работы: познакомиться со структурой программы на языке Ассемблер, разновидностями и назначением сегментов, способами организации простых и сложных типов данных, познакомиться со средствами создания программ на Ассемблере для ОС Linux.

Задание на лабораторную работу:

Задача: дан массив из 10 беззнаковых слов. Инвертировать биты старших байтов всех элементов массива. Найти сумму четных элементов полученного массива.

## 2 Ход Работы

Для начала выполнения лабораторной работы необходимо сначала обновить Ubuntu командой `apt update` и установить расширенный ассемблер `nasm` (рисунок 2.1).

```
german@german-VirtualBox:~$ sudo apt install nasm
Чтение списков пакетов... Готово
Построение дерева зависимостей
Чтение информации о состоянии... Готово
Следующие пакеты устанавливались автоматически и больше не требуются:
  linux-image-5.11.0-27-generic linux-modules-5.11.0-27-generic
  linux-modules-extra-5.11.0-27-generic
Для их удаления используйте «sudo apt autoremove».
Следующие НОВЫЕ пакеты будут установлены:
  nasm
Обновлено 0 пакетов, установлено 1 новых пакетов, для удаления отмечено 0 пакетов, и 164 пакетов не обновлено.
Необходимо скачать 362 кВ архивов.
После данной операции объем занятого дискового пространства возрастёт на 3 374 кВ.
Пол:1 http://ru.archive.ubuntu.com/ubuntu focal/universe amd64 nasm amd64 2.14.02-1 [362 кВ]
Получено 362 кВ за 1с (589 кВ/с)
Выбор ранее не выбранного пакета nasm.
(Чтение базы данных ... на данный момент установлено 227884 файла и каталога.)
Подготовка к распаковке .../nasm_2.14.02-1_amd64.deb ...
Распаковывается nasm (2.14.02-1) ...
Настраивается пакет nasm (2.14.02-1) ...
Обрабатываются триггеры для man-db (2.9.1-1) ...
german@german-VirtualBox:~$
```

Рисунок 2.1 – Установка Nasm

Далее через `nano` создадим файл, куда запишем код ассемблера для простой программы вывода текстовой строки (рисунок 2.2).

```
GNU nano 4.8 lab2test.asm Изменён
global _start

section .data
msg: db "Hello world!"
len equ $-msg

section .text
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg
mov edx, len
int 0x80

mov eax, 1
mov ebx, 0
int 0x80
```

Рисунок 2.2 – Код

Далее командой `nasm -f elf lab2test.asm` скомпилируем наш код, а командой `ls` проверим, чтобы появился объектный файл `lab2test.o` (рисунок 2.3).

```
german@german-VirtualBox:~$ nano lab2test.asm
german@german-VirtualBox:~$ nasm -f elf lab2test.asm
german@german-VirtualBox:~$ ls
bgk.sh          lab2test.asm    snap            Документы      Изображения    Общедоступные  Шаблоны
dockerfilebgk  lab2test.o      Видео           Загрузки       Музыка          'Рабочий стол'
```

Рисунок 2.3 – Компилирование файла и создание объектного файла

Далее скачаем и установим в системе компилятор gcc и отладчик gdb (рисунки 2.4-2.5).

```
german@german-VirtualBox:~$ sudo apt install gcc
[sudo] пароль для german:
Чтение списков пакетов... Готово
Построение дерева зависимостей
Чтение информации о состоянии... Готово
Уже установлен пакет gcc самой новой версии (4:9.3.0-1ubuntu2).
gcc помечен как установленный вручную.
Следующие пакеты устанавливались автоматически и больше не требуются:
  linux-image-5.11.0-27-generic linux-modules-5.11.0-27-generic linux-modules-extra-5.11.0-27-generic
Для их удаления используйте «sudo apt autoremove».
Обновлено 0 пакетов, установлено 0 новых пакетов, для удаления отмечено 0 пакетов, и 164 пакетов не обновлено.
german@german-VirtualBox:~$
```

Рисунок 2.4 – Установка gcc

```
german@german-VirtualBox:~$ sudo apt install gdb
[sudo] пароль для german:
Чтение списков пакетов... Готово
Построение дерева зависимостей
Чтение информации о состоянии... Готово
Следующие пакеты устанавливались автоматически и больше не требуются:
  linux-image-5.11.0-27-generic linux-modules-5.11.0-27-generic linux-modules-extra-5.11.0-27-generic
Для их удаления используйте «sudo apt autoremove».
Предлагаемые пакеты:
  gdb-doc
Следующие пакеты будут обновлены:
  gdb
Обновлено 1 пакетов, установлено 0 новых пакетов, для удаления отмечено 0 пакетов, и 163 пакетов не обновлено.
Необходимо скачать 0 B/3 222 kB архивов.
После данной операции объем занятого дискового пространства возрастет на 0 B.
(Чтение базы данных ... на данный момент установлено 227909 файлов и каталогов.)
Подготовка к распаковке .../gdb_9.2-0ubuntu1~20.04.1_amd64.deb ...
Распаковывается gdb (9.2-0ubuntu1~20.04.1) на замену (9.2-0ubuntu1~20.04) ...
Настраивается пакет gdb (9.2-0ubuntu1~20.04.1) ...
Обрабатываются триггеры для man-db (2.9.1-1) ...
german@german-VirtualBox:~$
```

Рисунок 2.5 – Установка gdb

Командой `ld -m elf_i386 lab2test.o -o lab2test` происходит связывание (рисунок 2.6).

```
Hello world!german@german-VirtualBox:~$ ld -m elf_i386 lab2test.o -o lab2test
german@german-VirtualBox:~$ ./lab2test
Hello world!german@german-VirtualBox:~$
```

Рисунок 2.6 – Связывание

Теперь напишем аналогичную программу, но на языке Си, и сделаем действия, описанные выше (рисунки 2.7-2.8).

```
german@german-VirtualBox: ~
GNU nano 4.8 lab2test.c
#include <stdio.h>

int main ()
{
    printf("hello world!");
    return 0;
}
```

Рисунок 2.7 – Код

```

german@german-VirtualBox:~$ nano lab2test.c
german@german-VirtualBox:~$ gcc lab2test.c -o lab2testC
german@german-VirtualBox:~$ ./lab2testC
hello world!german@german-VirtualBox:~$

```

Рисунок 2.8 – работа с компилятором

Далее напишем код на ассемблере согласно индивидуальному заданию (рисунок 2.9) и воспользуемся отладчиком gdb. Командой `b` задаётся точка останова, команда `r` запускает отладку, команда `layout reg` показывает информацию о регистрах во время работы. Соответственно сумма элементов массива будет храниться в регистре `ebx` (рисунки 2.10-2.11)

```

GNU nano 4.8                                lab2zadanie.asm
global _start

section .data
massiv1 dw 0x45, 0x4000, 0xF, 0x1, 0x2, 0x9, 0x1, 0x7, 0x123, 0xA

section .bss
massiv2 resw 10

section .text
_start:
mov ecx, 10
mov esi, massiv1
mov edi, massiv2
xor eax, eax

loop1:
lodsw
not ah
stosw
loop loop1
mov ecx, 10
mov esi, massiv2
xor ebx, ebx

loop2:
lodsw
test eax, 1
jnp nechet
add ebx, eax

nechet:
cmp ecx, 0
je exit
loop loop2

exit:
mov eax, 1
mov ebx, 0
int 0x80

```

Рисунок 2.9 – Код

```

german@german-VirtualBox:~$ nano lab2zadanie.asm
german@german-VirtualBox:~$ nasm -f elf lab2zadanie.asm
german@german-VirtualBox:~$ ld -m elf_i386 lab2zadanie.o -o lab2zadanie
german@german-VirtualBox:~$ gdb lab2zadanie
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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 lab2zadanie...
(No debugging symbols found in lab2zadanie)
(gdb) b _start
Breakpoint 1 at 0x08049000
(gdb) r
Starting program: /home/german/lab2zadanie

Breakpoint 1, 0x08049000 in _start ()
(gdb) layout reg

```

Рисунок 2.10 – Работа с отладчиком

Register group: general					
eax	0xffff0a	65290	ecx	0x0	0
edx	0x0	0	ebx	0x2bd0c	179468
esp	0xfffffd270	0xfffffd270	ebp	0x0	0x0
esi	0x804a028	134520872	edi	0x804a028	134520872
eip	0x8049037	0x8049037 <exit>	eflags	0x202	[ IF ]
cs	0x23	35	ss	0x2b	43
ds	0x2b	43	es	0x2b	43
fs	0x0	0	gs	0x0	0

  

```

B+ 0x8049000 <_start>    mov     $0xa,%ecx
0x8049005 <_start+5>    mov     $0x804a000,%esi
0x804900a <_start+10>   mov     $0x804a014,%edi
0x804900f <_start+15>   xor     %eax,%eax
0x8049011 <loop1>       lods     %ds:(%esi),%ax
0x8049013 <loop1+2>     not     %ah
0x8049015 <loop1+4>     stos     %ax,%es:(%edi)
0x8049017 <loop1+6>     loop    0x8049011 <loop1>
0x8049019 <loop1+8>     mov     $0xa,%ecx
0x804901e <loop1+13>    mov     $0x804a014,%esi
0x8049023 <loop1+18>    xor     %ebx,%ebx
0x8049025 <loop2>       lods     %ds:(%esi),%ax
0x8049027 <loop2+2>     test    $0x1,%eax
0x804902c <loop2+7>     jnp     0x8049030 <nechet>

```

native No process in: L?? PC: ??

```

(gdb) s
Single stepping until exit from function loop2,
which has no line number information.
0x08049030 in nechet ()
(gdb) s
Single stepping until exit from function nechet,
which has no line number information.
0x08049037 in exit ()
(gdb) s
Single stepping until exit from function exit,
which has no line number information.
[Inferior 1 (process 32426) exited normally]
(gdb) s
The program is not being run.
(gdb)

```

Рисунок 2.11 – Работа с отладчиком

После того, как мы проверили работоспособность программы через компилятор, нужно дизассемблировать код, для этого нужно ввести команду: `objdump -d -M i386 -M intel-mnemonic lab2zadanie` (рисунок 2.12).

```
german@german-VirtualBox:~$ objdump -d lab2zadanie
lab2zadanie:      формат файла elf32-i386

Дизассемблирование раздела .text:

08049000 <_start>:
08049000:  b9 0a 00 00 00      mov     $0xa,%ecx
08049005:  be 00 a0 04 08      mov     $0x804a000,%esi
0804900a:  bf 14 a0 04 08      mov     $0x804a014,%edi
0804900f:  31 c0               xor     %eax,%eax

08049011 <loop1>:
08049011:  66 ad               lods    %ds:(%esi),%ax
08049013:  f6 d4               not     %ah
08049015:  66 ab               stos    %ax,%es:(%edi)
08049017:  e2 f8               loop    08049011 <loop1>
08049019:  b9 0a 00 00 00      mov     $0xa,%ecx
0804901e:  be 14 a0 04 08      mov     $0x804a014,%esi
08049023:  31 db               xor     %ebx,%ebx

08049025 <loop2>:
08049025:  66 ad               lods    %ds:(%esi),%ax
08049027:  a9 01 00 00 00      test    $0x1,%eax
0804902c:  7b 02               jnp     08049030 <nechet>
0804902e:  01 c3               add     %eax,%ebx

08049030 <nechet>:
08049030:  83 f9 00            cmp     $0x0,%ecx
08049033:  74 02               je      08049037 <exit>
08049035:  e2 ee               loop    08049025 <loop2>

08049037 <exit>:
08049037:  b8 01 00 00 00      mov     $0x1,%eax
0804903c:  bb 00 00 00 00      mov     $0x0,%ebx
08049041:  cd 80               int     $0x80
```

Рисунок 2.12 – Дизассемблирование кода

После того как сделали всю работу на ассемблере, необходимо написать программу на языке C, запустить через компилятор gcc и дизассемблировать. В итоге, сравнить какой код выполняется быстрее (рисунки 2.13 – 2.15).

```
GNU nano 4.8                                zadanie2labC.c
#include "stdint.h"
#include <stdio.h>
uint16_t function(uint16_t x)
{
    uint16_t l=0x00ff&x;
    uint16_t h=0xff00&x;
    h=~h;
    h=0xff00&h;
    h+=l;
    return h;
}

int main()
{
    uint16_t massiv[10]={0x45, 0x4000, 0xF, 0x1, 0x2, 0x9, 0x1, 0x7, 0x123, 0xA};
    uint32_t summ=0;
    for(int i=0;i<10;i++)
    {
        massiv[i]=function(massiv[i]);
        if((massiv[i]%2)==0)
        {
            printf("%x\n", massiv[i]);
            summ +=massiv[i];
        }
    }
    printf("\n%x\n", summ);
    return 0;
}
```

Рисунок 2.13 – Код на Си

```

german@german-VirtualBox:~$ nano zadanie2labC.c
german@german-VirtualBox:~$ gcc zadanie2labC.c -o zadanie2labC
german@german-VirtualBox:~$ ./zadanie2labC
bf00
ff02
ff0a

2bd0c
german@german-VirtualBox:~$ █

```

Рисунок 2.14 – Работа с компилятором

```

german@german-VirtualBox:~$ objdump -d zadanie2labC

zadanie2labC:    формат файла elf64-x86-64

Дизассемблирование раздела .init:
0000000000001000 <.init>:
1000:    f3 0f 1e fa            endbr64
1004:    48 83 ec 08            sub    $0x8,%rsp
1008:    48 8b 05 d9 2f 00 00    mov    0x2fd9(%rip),%rax        # 3fe8 <__gmon_start__>
100f:    48 85 c0                test   %rax,%rax
1012:    74 02                  je     1016 <_init+0x16>
1014:    ff d0                  callq  *%rax
1016:    48 83 c4 08            add    $0x8,%rsp
101a:    c3                     retq

Дизассемблирование раздела .plt:
0000000000001020 <.plt>:
1020:    ff 35 92 2f 00 00      pushq  0x2f92(%rip)            # 3fb8 <_GLOBAL_OFFSET_TABLE_+0x8>
1026:    f2 ff 25 93 2f 00 00    bnd jmpq *0x2f93(%rip)        # 3fc0 <_GLOBAL_OFFSET_TABLE_+0x10>
102d:    0f 1f 00                nopl   (%rax)
1030:    f3 0f 1e fa            endbr64
1034:    68 00 00 00 00          pushq  $0x0
1039:    f2 e9 e1 ff ff ff      bnd jmpq 1020 <.plt>
103f:    90                      nop
1040:    f3 0f 1e fa            endbr64
1044:    68 01 00 00 00          pushq  $0x1
1049:    f2 e9 d1 ff ff ff      bnd jmpq 1020 <.plt>
104f:    90                      nop

Дизассемблирование раздела .plt.got:
0000000000001050 <_cxa_finalize@plt>:
1050:    f3 0f 1e fa            endbr64
1054:    f2 ff 25 9d 2f 00 00    bnd jmpq *0x2f9d(%rip)        # 3ff8 <_cxa_finalize@GLIBC_2.2.5>
105b:    0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

Дизассемблирование раздела .plt.sec:
0000000000001060 <__stack_chk_fail@plt>:
1060:    f3 0f 1e fa            endbr64
1064:    f2 ff 25 5d 2f 00 00    bnd jmpq *0x2f5d(%rip)        # 3fc8 <__stack_chk_fail@GLIBC_2.4>
106b:    0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

```

Рисунок 2.15 – Дизассемблирование кода

Теперь проверим сколько занимают места оба исполняемых файла и сколько времени они выполняются (рисунки 2.16-2.17).

```

german@german-VirtualBox:~$ du --bytes zadanie2labC
16792  zadanie2labC
german@german-VirtualBox:~$ du --bytes lab2zadanie
8860   lab2zadanie
german@german-VirtualBox:~$ du --bytes zadanie2labC █

```

Рисунок 2.16 – Дисковое пространство



```

german@german-VirtualBox:~$ time ./zadanie2labC
bf00
ff02
ff0a

2bd0c

real    0m0,002s
user    0m0,001s
sys     0m0,001s
german@german-VirtualBox:~$ time ./lab2zadanie

real    0m0,002s
user    0m0,000s
sys     0m0,001s

```

Рисунок 2.17 – Время работы

Теперь соберем докер контейнер (рисунок 2.18).

```

german@german-VirtualBox:~$ docker build -t lab2 -f dockerfilebgk .
Sending build context to Docker daemon 151.8MB
Step 1/7 : FROM ubuntu
--> 2b4cba85892a
Step 2/7 : RUN apt update
--> Using cache
--> 7d526d8d0431
Step 3/7 : RUN apt install gcc -y
--> Using cache
--> f55da3636fca
Step 4/7 : RUN apt install nano -y
--> Using cache
--> 0edf856e2718
Step 5/7 : COPY zadanie2labC.c .
--> c2cf56065029
Step 6/7 : RUN gcc zadanie2labC.c -o zadanie2labC
--> Running in fa85179ead50
Removing intermediate container fa85179ead50
--> 18fc87c402ed
Step 7/7 : CMD ./zadanie2labC
--> Running in 41e35cb018fd
Removing intermediate container 41e35cb018fd
--> 0442e0b7b228
Successfully built 0442e0b7b228
Successfully tagged lab2:latest
german@german-VirtualBox:~$ docker run -it 0442e0b7b228
bf00
ff02
ff0a

2bd0c
german@german-VirtualBox:~$

```

Рисунок 2.18 – Сборка

Ссылка на github: <https://github.com/CoReShAdA/>

### 3 Заключение

В ходе работы были написаны программы на языке высокого уровня и ассемблера, выполняющие одинаковый функционал.