

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

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ASSEMBLER

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

Вариант 2

Студент гр. 727-1

_____ Антипов Р.С.

____.____.2020

Преподаватель кафедры

КИБЭВС

_____ Полюга В.А.

____.____.2020

1 Введение

Целью работы является ознакомление с основами программирования на языке assembler и средствами создания программ для linux. Написать программу на языке assembler и C, выполняющую следующую задачу: дан массив из 12 беззнаковых чисел (байтов). Определить количество тех элементов массива, двоичные коды которых содержат 0 в битах 1 и 5.

2 Ход работы

Для выполнения индивидуального задания был использован ассемблер NASM и Си, а также компилятор gcc, дебаггер gdb и дизассемблер objdump. Была реализована программа на NASM:

section .data

array db 35, 12, 64, 23, 101, 68, 89, 53, 27, 95, 3, 16; входной массив
array_size equ 12; размер массива
mask equ 17; маска 010001
count db 0
output db '%i', 0xA, 0x0

section .text

global main; объявление точки входа
extern printf; функция вывода из библиотеки

;точка входа

main:

mov esi, array; ссылка на массив
mov ecx, array_size; размер массива
xor eax, eax; зануление eax
xor ebx, ebx; зануление ebx

;проход по элементам массива

walkthrough:

lodsb; читает один байт из регистра esi в регистр eax

xor eax, mask

and eax, mask

cmp eax, mask; если после выполнения предыдущих операций xor и and выполняется равенство, то число подходит

JNE false; если равенство не выполняется, переход на метку false

inc ebx; если равенство выполняется, то увеличиваем значение ebx на 1

;переход к следующему элементу массива

false:

loop walkthrough; уменьшает на 1 ecx, если после этого он не равен 0, то переход на метку walkthrough

exit:

push ebx

push output

call printf

add esp, 8

mov eax, 1;

mov ebx, 0

int 0x80

Создание исполняемого файла и проверка работы программы (рисунок 2.1).



```

rodion@rodion-pc: /media/rodion/Drive/Учеба/универ/8 семестр/СП/Lab2$ nasm -s -f
elf lab2_noprint.asm; ld -S -m elf_i386 -o lab2_noprint lab2_noprint.o
ld: предупреждение: невозможно найти символ входа _start; используем значение по
умолчанию 0000000008049000
rodion@rodion-pc: /media/rodion/Drive/Учеба/универ/8 семестр/СП/Lab2$ ./lab2_nopr
int
rodion@rodion-pc: /media/rodion/Drive/Учеба/универ/8 семестр/СП/Lab2$

```

Рисунок 2.1 – Создание исполняемого файла и запуск программы на ассемблере

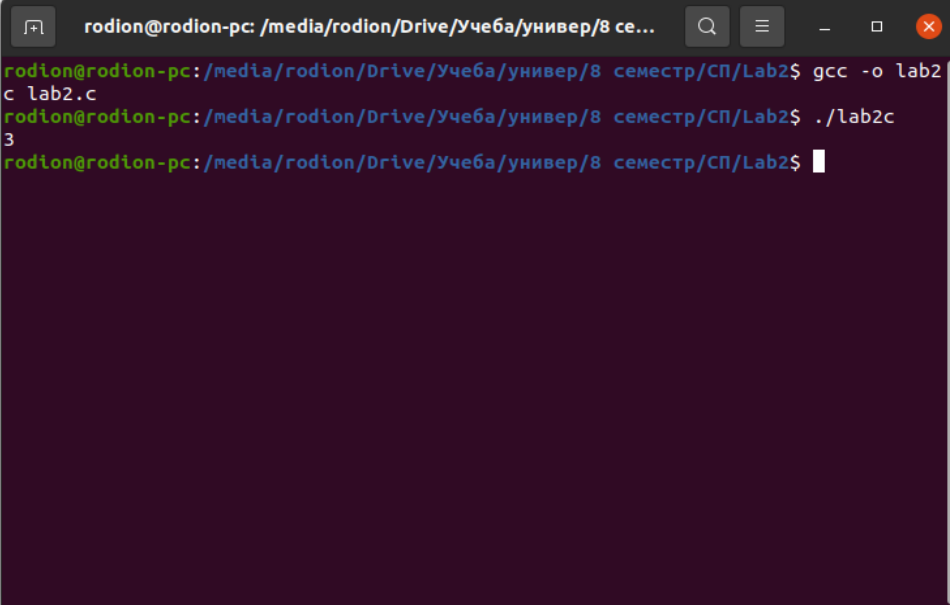
Была реализована программа на Си:

```

#include <stdio.h>
int main(void)
{
    int array[12] = {35, 12, 64, 23, 101, 68, 89, 53, 27, 95, 3, 16};
    int count = 0;
    int mask = 17;
    for (int i = 0; i < 12; i++)
    {
        array[i] = array[i] ^ mask;
        array[i] = array[i] & mask;
        if (array[i] == mask)
            count++;
    }
    printf("%i\n", count);
    return 0;
}

```

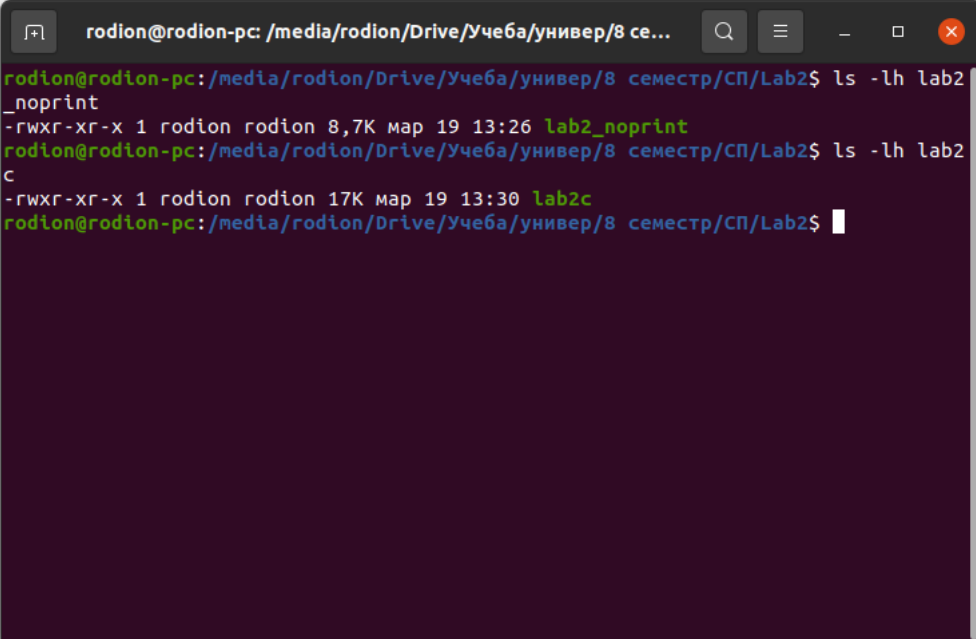
Создание исполняемого файла и проверка работы программы (рисунок 2.2).



```
rodion@rodion-pc: /media/rodion/Drive/Учеба/универ/8 се...
rodion@rodion-pc:/media/rodion/Drive/Учеба/универ/8 семестр/СП/Lab2$ gcc -o lab2
c lab2.c
rodion@rodion-pc:/media/rodion/Drive/Учеба/универ/8 семестр/СП/Lab2$ ./lab2c
3
rodion@rodion-pc:/media/rodion/Drive/Учеба/универ/8 семестр/СП/Lab2$
```

Рисунок 2.2 – Создание исполняемого файла и запуск программы на Си

В результате получились исполняемые файлы разного размера. Исполняемый файл на языке ассемблера имеет значительно меньший размер (рисунок 2.3).



```
rodion@rodion-pc: /media/rodion/Drive/Учеба/универ/8 се...
rodion@rodion-pc:/media/rodion/Drive/Учеба/универ/8 семестр/СП/Lab2$ ls -lh lab2
_noprint
-rwxr-xr-x 1 rodion rodion 8,7K map 19 13:26 lab2_noprint
rodion@rodion-pc:/media/rodion/Drive/Учеба/универ/8 семестр/СП/Lab2$ ls -lh lab2
c
-rwxr-xr-x 1 rodion rodion 17K map 19 13:30 lab2c
rodion@rodion-pc:/media/rodion/Drive/Учеба/универ/8 семестр/СП/Lab2$
```

Рисунок 2.3 — Сравнение размеров исполняемых файлов

Обе программы были дизассемблированы. Далее приведен дизассемблированный код программы на Си:

Dump of assembler code for function main:

```
=> 0x000055555555169 <+0>:    endbr64
0x00005555555516d <+4>:    push    rbp
0x00005555555516e <+5>:    mov     rbp, rsp
0x000055555555171 <+8>:    sub     rsp, 0x50
0x000055555555175 <+12>:   mov     rax, QWORD PTR fs:0x28
0x00005555555517e <+21>:   mov     QWORD PTR [rbp-0x8], rax
0x000055555555182 <+25>:   xor     eax, eax
0x000055555555184 <+27>:   mov     DWORD PTR [rbp-0x40], 0x23
0x00005555555518b <+34>:   mov     DWORD PTR [rbp-0x3c], 0xc
0x000055555555192 <+41>:   mov     DWORD PTR [rbp-0x38], 0x40
0x000055555555199 <+48>:   mov     DWORD PTR [rbp-0x34], 0x17
0x0000555555551a0 <+55>:   mov     DWORD PTR [rbp-0x30], 0x65
0x0000555555551a7 <+62>:   mov     DWORD PTR [rbp-0x2c], 0x44
0x0000555555551ae <+69>:   mov     DWORD PTR [rbp-0x28], 0x59
0x0000555555551b5 <+76>:   mov     DWORD PTR [rbp-0x24], 0x35
0x0000555555551bc <+83>:   mov     DWORD PTR [rbp-0x20], 0x1b
0x0000555555551c3 <+90>:   mov     DWORD PTR [rbp-0x1c], 0x5f
0x0000555555551ca <+97>:   mov     DWORD PTR [rbp-0x18], 0x3
0x0000555555551d1 <+104>:  mov     DWORD PTR [rbp-0x14], 0x10
0x0000555555551d8 <+111>:  mov     DWORD PTR [rbp-0x4c], 0x0
0x0000555555551df <+118>:  mov     DWORD PTR [rbp-0x44], 0x11
0x0000555555551e6 <+125>:  mov     DWORD PTR [rbp-0x48], 0x0
0x0000555555551ed <+132>:  jmp     0x55555555233 <main+202>
0x0000555555551ef <+134>:  mov     eax, DWORD PTR [rbp-0x48]
0x0000555555551f2 <+137>:  cdqe
0x0000555555551f4 <+139>:  mov     eax, DWORD PTR [rbp+rax*4-0x40]
0x0000555555551f8 <+143>:  xor     eax, DWORD PTR [rbp-0x44]
0x0000555555551fb <+146>:  mov     edx, eax
0x0000555555551fd <+148>:  mov     eax, DWORD PTR [rbp-0x48]
0x000055555555200 <+151>:  cdqe
0x000055555555202 <+153>:  mov     DWORD PTR [rbp+rax*4-0x40], edx
0x000055555555206 <+157>:  mov     eax, DWORD PTR [rbp-0x48]
0x000055555555209 <+160>:  cdqe
0x00005555555520b <+162>:  mov     eax, DWORD PTR [rbp+rax*4-0x40]
0x00005555555520f <+166>:  and     eax, DWORD PTR [rbp-0x44]
0x000055555555212 <+169>:  mov     edx, eax
0x000055555555214 <+171>:  mov     eax, DWORD PTR [rbp-0x48]
0x000055555555217 <+174>:  cdqe
0x000055555555219 <+176>:  mov     DWORD PTR [rbp+rax*4-0x40], edx
0x00005555555521d <+180>:  mov     eax, DWORD PTR [rbp-0x48]
0x000055555555220 <+183>:  cdqe
0x000055555555222 <+185>:  mov     eax, DWORD PTR [rbp+rax*4-0x40]
0x000055555555226 <+189>:  cmp     DWORD PTR [rbp-0x44], eax
0x000055555555229 <+192>:  jne     0x5555555522f <main+198>
0x00005555555522b <+194>:  add     DWORD PTR [rbp-0x4c], 0x1
0x00005555555522f <+198>:  add     DWORD PTR [rbp-0x48], 0x1
0x000055555555233 <+202>:  cmp     DWORD PTR [rbp-0x48], 0xb
```

```

0x000055555555237 <+206>: jle 0x555555551ef <main+134>
0x000055555555239 <+208>: mov  eax,DWORD PTR [rbp-0x4c]
0x00005555555523c <+211>: mov  esi,eax
0x00005555555523e <+213>: lea  rdi,[rip+0xdbf]      # 0x555555556004
0x000055555555245 <+220>: mov  eax,0x0
0x00005555555524a <+225>: call 0x55555555070 <printf@plt>
0x00005555555524f <+230>: mov  eax,0x0
0x000055555555254 <+235>: mov  rcx,QWORD PTR [rbp-0x8]
0x000055555555258 <+239>: xor  rcx,QWORD PTR fs:0x28
0x000055555555261 <+248>: je   0x55555555268 <main+255>
0x000055555555263 <+250>: call 0x55555555060 <__stack_chk_fail@plt>
0x000055555555268 <+255>: leave
0x000055555555269 <+256>: ret

```

Далее приведен дизассемблированный код программы на ассемблере:

Дизассемблирование раздела .text:

```

08049000 <main>:
08049000:  be 00 a0 04 08      mov  esi,0x804a000
08049005:  b9 0c 00 00 00      mov  ecx,0xc
0804900a:  31 c0               xor  eax,eax
0804900c:  31 db               xor  ebx,ebx

0804900e <walkthrough>:
0804900e:  ac                 lods  al,BYTE PTR ds:[esi]
0804900f:  83 f0 11           xor  eax,0x11
08049012:  83 e0 11           and  eax,0x11
08049015:  83 f8 11           cmp  eax,0x11
08049018:  75 01             jne  804901b <false>
0804901a:  43                 inc  ebx

0804901b <false>:
0804901b:  e2 f1             loop 804900e <walkthrough>

0804901d <exit>:
0804901d:  b8 01 00 00 00      mov  eax,0x1
08049022:  bb 00 00 00 00      mov  ebx,0x0
08049027:  cd 80             int  0x80

```

Дизассемблирование раздела .data:

```

0804a000 <array>:
0804a000:  23 0c 40           and  ecx,DWORD PTR [eax+eax*2]
0804a003:  17                 pop  ss
0804a004:  65 44             gs inc esp
0804a006:  59                 pop  ecx
0804a007:  35 1b 5f 03 10     xor  eax,0x10035f1b

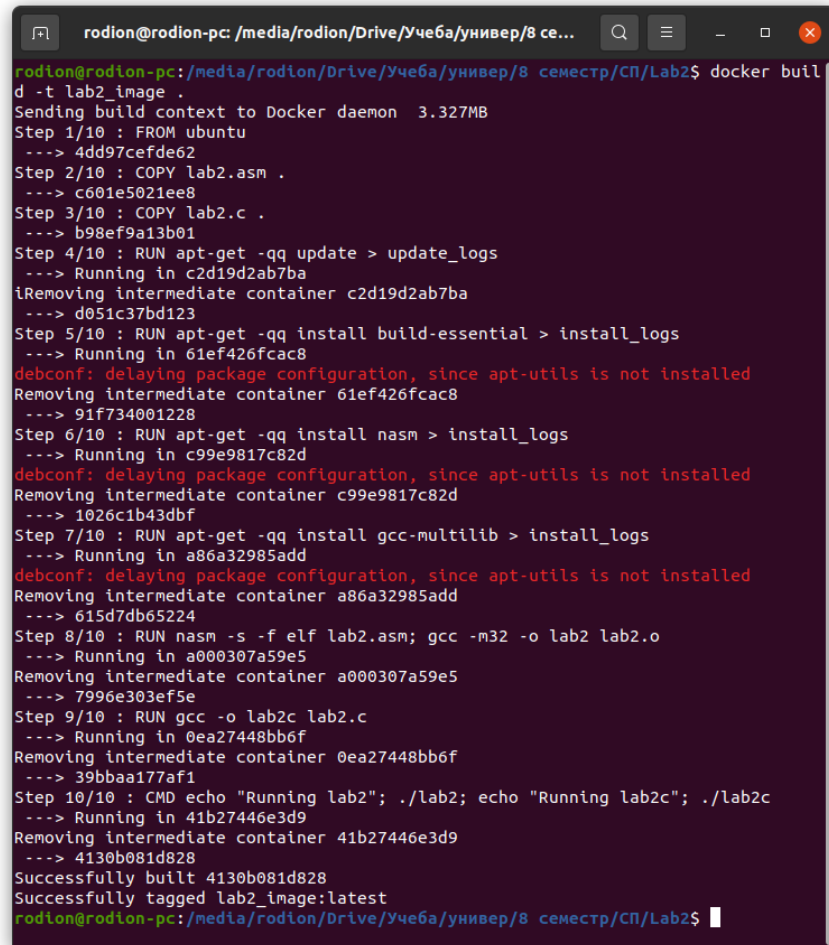
0804a00c <count>:
...

```

0804a00d <output>:

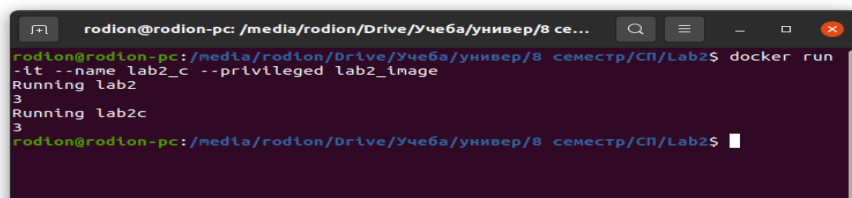
```
804a00d:    25             .byte 0x25
804a00e:    69             .byte 0x69
804a00f:    0a 00         or    al,BYTE PTR [eax]
```

На основе образа Ubuntu был создан docker контейнер, который создает исполняемые файлы этих программ и запускает их (рисунки 2.4 — 2.5).



```
rodion@rodion-pc: /media/rodion/Drive/Учеба/универ/8 семестр/СН/Lab2$ docker build
d -t lab2_image .
Sending build context to Docker daemon 3.327MB
Step 1/10 : FROM ubuntu
--> 4dd97cefde62
Step 2/10 : COPY lab2.asm .
--> c601e5021ee8
Step 3/10 : COPY lab2.c .
--> b98ef9a13b01
Step 4/10 : RUN apt-get -qq update > update_logs
--> Running in c2d19d2ab7ba
Removing intermediate container c2d19d2ab7ba
--> d051c37bd123
Step 5/10 : RUN apt-get -qq install build-essential > install_logs
--> Running in 61ef426fcac8
debconf: delaying package configuration, since apt-utils is not installed
Removing intermediate container 61ef426fcac8
--> 91f734001228
Step 6/10 : RUN apt-get -qq install nasm > install_logs
--> Running in c99e9817c82d
debconf: delaying package configuration, since apt-utils is not installed
Removing intermediate container c99e9817c82d
--> 1026c1b43dbf
Step 7/10 : RUN apt-get -qq install gcc-multilib > install_logs
--> Running in a86a32985add
debconf: delaying package configuration, since apt-utils is not installed
Removing intermediate container a86a32985add
--> 615d7db65224
Step 8/10 : RUN nasm -s -f elf lab2.asm; gcc -m32 -o lab2 lab2.o
--> Running in a000307a59e5
Removing intermediate container a000307a59e5
--> 7996e303ef5e
Step 9/10 : RUN gcc -o lab2c lab2.c
--> Running in 0ea27448bb6f
Removing intermediate container 0ea27448bb6f
--> 39bbaa177af1
Step 10/10 : CMD echo "Running lab2"; ./lab2; echo "Running lab2c"; ./lab2c
--> Running in 41b27446e3d9
Removing intermediate container 41b27446e3d9
--> 4130b081d828
Successfully built 4130b081d828
Successfully tagged lab2_image:latest
rodion@rodion-pc: /media/rodion/Drive/Учеба/универ/8 семестр/СН/Lab2$
```

Рисунок 2.4 — Создание образа



```
rodion@rodion-pc: /media/rodion/Drive/Учеба/универ/8 семестр/СН/Lab2$ docker run
-it --name lab2_c --privileged lab2_image
Running lab2
3
Running lab2c
3
rodion@rodion-pc: /media/rodion/Drive/Учеба/универ/8 семестр/СН/Lab2$
```

Рисунок 2.5 — Запуск контейнера

3 Заключение

В результате выполнения лабораторной работы было произведено ознакомление с основами программирования на языке assembler и средствами создания программ для linux. Написана программа на языке assembler и C, выполняющую следующую задачу: дан массив из 12 беззнаковых чисел (байтов). Определить количество тех элементов массива, двоичные коды которых содержат 0 в битах 1 и 5.