Министерство образования и науки РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Санкт-Петербургский национальный исследовательский университет

ИТМО»

**факультет безопасности информационных технологий**

**ЛАБОРАТОРНАЯ РАБОТА № 2**

Работу выполнил

студент

группы N3149

очного отделения

Шарифуллин И.А.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверено преподавателем

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Грозов В. А.)

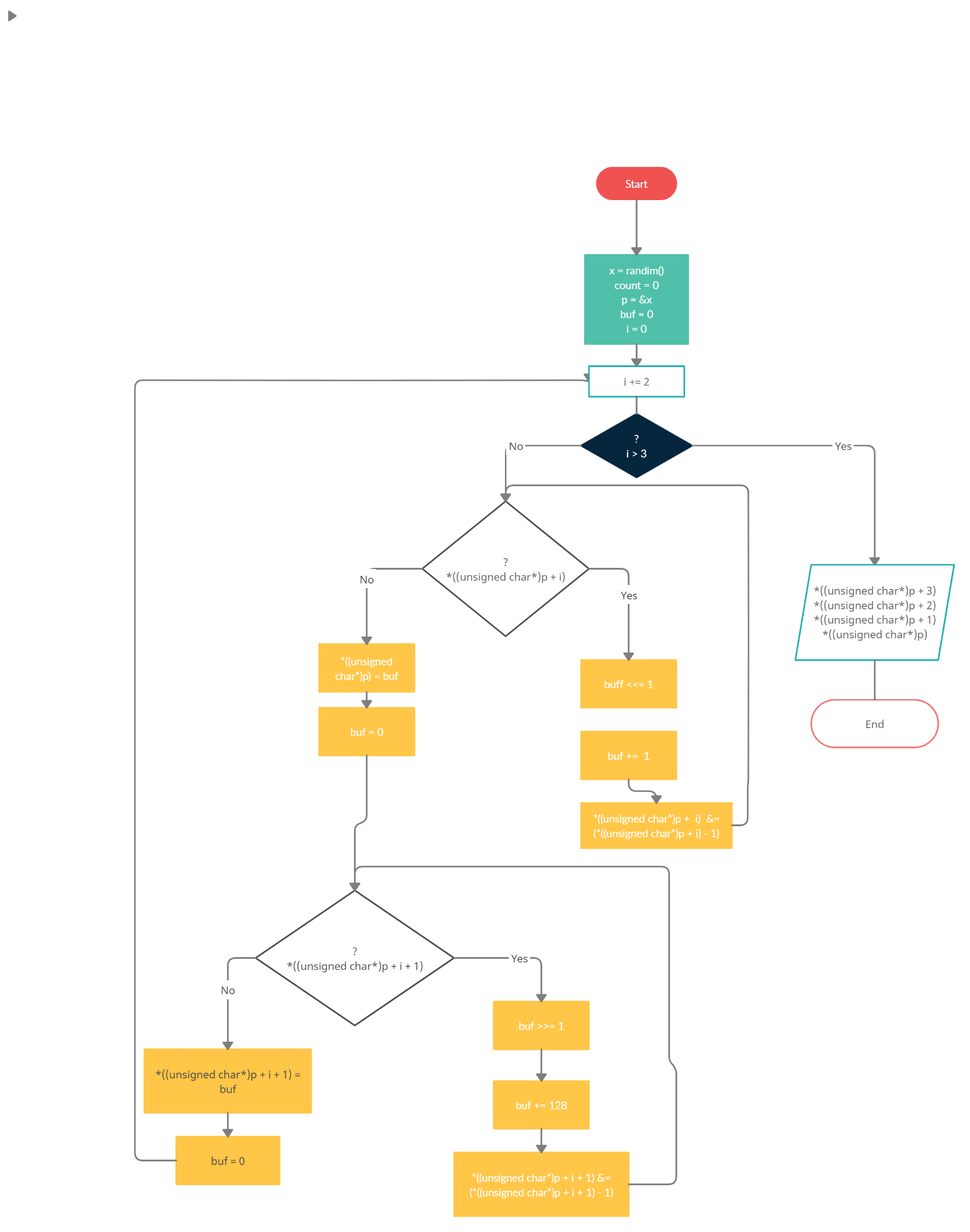


Санкт-Петербург, 2020

1. Задание:



1. Блок-схема алгоритма преобразования для программы на С:



1. Текст программы на С с комментариями:

#include <stdio.h>

void out(unsigned char a);

int main() {

int x = 0xFFF0F0F0;

int\* p = &x; //инициализируем указатель на наше число

int count = 0;

unsigned char buf = 0;

out(\*((unsigned char\*)p)); //Процедура вывода одного байта в двоичном виде

out(\*((unsigned char\*)p + 1));

out(\*((unsigned char\*)p + 2));

out(\*((unsigned char\*)p + 3));

printf("\n");

for (int i = 0; i < 3; i += 2) { //Цикл обрабатывает 2 байта за проход (2 прохода)

while (\*((unsigned char\*)p + i)) { //пока байт числа не равен нулю

buf <<= 1; //сдвигаем биты на один влево в буфере

buf += 1; //ставим 1 в самый правый бит буфера

\*((unsigned char\*)p + i) &= (\*((unsigned char\*)p + i) - 1); //удаляем последнюю единицу байта

}

\*((unsigned char\*)p + i) = buf; //Заносим готовый байт обратно в число

buf = 0; //Обнуляем буффер

while (\*((unsigned char\*)p + i + 1)) { //пока следующий байт числа не равен нулю

buf >>= 1; //сдвигаем биты на один вправо в буфере

buf += 128; //ставим 1 в самый левый бит буфера

\*((unsigned char\*)p + i + 1) &= (\*((unsigned char\*)p + i + 1) - 1); //удаляем последнюю единицу байта

}

\*((unsigned char\*)p + i + 1) = buf; //Заносим готовый байт обратно в число

buf = 0; //Обнуляем буффер

}

out(\*((unsigned char\*)p)); //Выводим обработанное число

out(\*((unsigned char\*)p + 1));

out(\*((unsigned char\*)p + 2));

out(\*((unsigned char\*)p + 3));

return 0;

}

void out(unsigned char a) { //Процедура вывода одного байта в двоичном виде

char mas[8] = {};

int i = 0;

for (int cnt = 8; cnt > 0; cnt--) {

if (a & 1) //Кладем в массив число в двоичном виде в порядке, в котором оно хранится в памяти

mas[i] = 1;

else

mas[i] = 0;

i++;

a >>= 1;

}

for (int cnt = 7; cnt != -1; cnt--) { //Выводим содержимое массива в обратном порядке

if (mas[cnt])

printf("1");

else

printf("0");

}

printf(" "); }

1. Текст программы на ассемблере NASM с комментариями:

section .data

null dw '0' (db – 1 byte, dw – 2 byte, dd – 4 byte, dq – 8 byte)

one dw '1'

len equ $ - one (equ - const)

msg db 10

msg\_len equ $ - msg

section .text

global \_start

\_start: rdrand r14 ;генерируем случайное число

mov r8w, 32

mov r9, r14

jmp lp1 ;переходим на вывод числа

;XX 00 00 00

lp4: mov rcx, r14 ;берем первый слева байт числа

shr rcx, 24

xor rbx, rbx

xor rdx, rdx

lp5: shr bl, 1 ;сдвигаем буфер на 1 бит вправо

add bl, 128 ;добавляем единичку в самый левый бит байта буфера

mov dl, cl

dec dl

and cl, dl ;удаляем самую правую единицу байта числа

cmp cl, 0

jnz lp5

mov rsi, 256 ;сдвигаем ответ на 1 байт вправо

mul rsi

mov al, bl ;кладем в правый байт ответа наш обработанный байт

;00 XX 00 00 ;далее по аналогии работаем с 3 оставшимися байтами

mov rcx, r14

shr rcx, 16

xor rbx, rbx

xor rdx, rdx

lp6: shl bl, 1

inc bl

mov dl, cl

dec dl

and cl, dl

cmp cl, 0

jnz lp6

mov rsi, 256

mul rsi

mov al, bl

;00 00 XX 00

mov rcx, r14

shr rcx, 8

xor rbx, rbx

xor rdx, rdx

lp7: shr bl, 1

add bl, 128

mov dl, cl

dec dl

and cl, dl

cmp cl, 0

jnz lp7

mov rsi, 256

mul rsi

mov al, bl

;00 00 00 XX

mov rcx, r14

xor rbx, rbx

xor rdx, rdx

lp8: shl bl, 1

inc bl

mov dl, cl

dec dl

and cl, dl

cmp cl, 0

jnz lp8

mov rsi, 256

mul rsi

mov al, bl

mov r13, 1

mov r8w, 32

mov r9, rax

lp1: mov r10d,r9d ;двоичный вывод

and r10d, 10000000000000000000000000000000b

mov eax,4

mov ebx,1

mov edx, len

cmp r10d,0

jnz v

jz nv

v: mov ecx, one

int 80h

jmp ob3

nv: mov ecx, null

cmp r8w,0

int 80h

ob3: shl r9, 1

dec r8w

jnz lp1

mov rax, 1 ;перенос строки

mov rdi, 1

mov rsi, msg

mov rdx, msg\_len

syscall

cmp r13, 0

jz lp4

mov eax,1 ;выход из программы

mov ebx,0

int 80h

5. Дизассемблерный листинг существенных частей программы на С:

main:

sal BYTE PTR -29[rbp] ;сдвиг влево на 1 бит

add BYTE PTR -29[rbp], 1 ;добавление единички в самый правый бит буфера

mov eax, DWORD PTR -24[rbp]

movsx rdx, eax

mov rax, QWORD PTR -16[rbp]

add rax, rdx

movzx eax, BYTE PTR [rax]

mov ecx, eax

mov eax, DWORD PTR -24[rbp]

movsx rdx, eax

mov rax, QWORD PTR -16[rbp]

add rax, rdx

movzx eax, BYTE PTR [rax]

sub eax, 1

and ecx, eax ;удаление последней единички байта числа

mov eax, DWORD PTR -24[rbp]

movsx rdx, eax

mov rax, QWORD PTR -16[rbp]

add rax, rdx

mov edx, ecx

mov BYTE PTR [rax], dl

mov eax, DWORD PTR -24[rbp]

movsx rdx, eax

mov rax, QWORD PTR -16[rbp]

add rax, rdx

movzx eax, BYTE PTR [rax]

test al, al

jne .L4

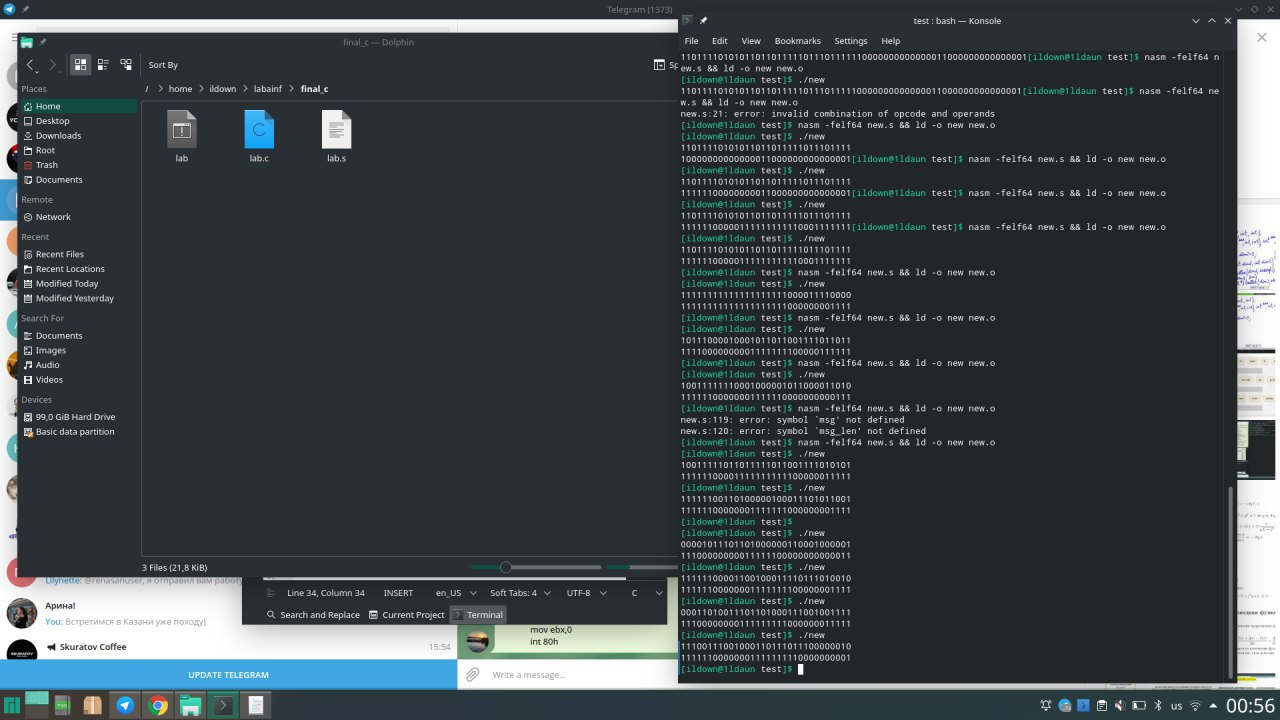
5. Краткий анализ по результатам сравнения программы на ассемблере и

дизассемблированной программы на С:

В листинге гораздо больше команд перемещения чисел. Тем не менее в нем есть тот же алгоритм, что и у меня в NASM. Это последовательный перенос единичек байта путем удаления их из числа и добавлением в буфер. А многочисленные операции перемещения, как я полагаю, позволяют компилятору добиться хорошей скорости работы.

1. Скриншоты прогонов программ на различных исходных данных:

Результат прогонов программы, написанной на NASM:



Результат прогонов программы, написанной на C:

