

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

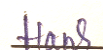
**«Национальный исследовательский университет ИТМО»**

**Факультет безопасности информационных технологий**


**Дисциплина: «Информатика»**

## **ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

Выполнил: Студент  
группы N3149

 Нгуен Хонг Хань

Проверил:

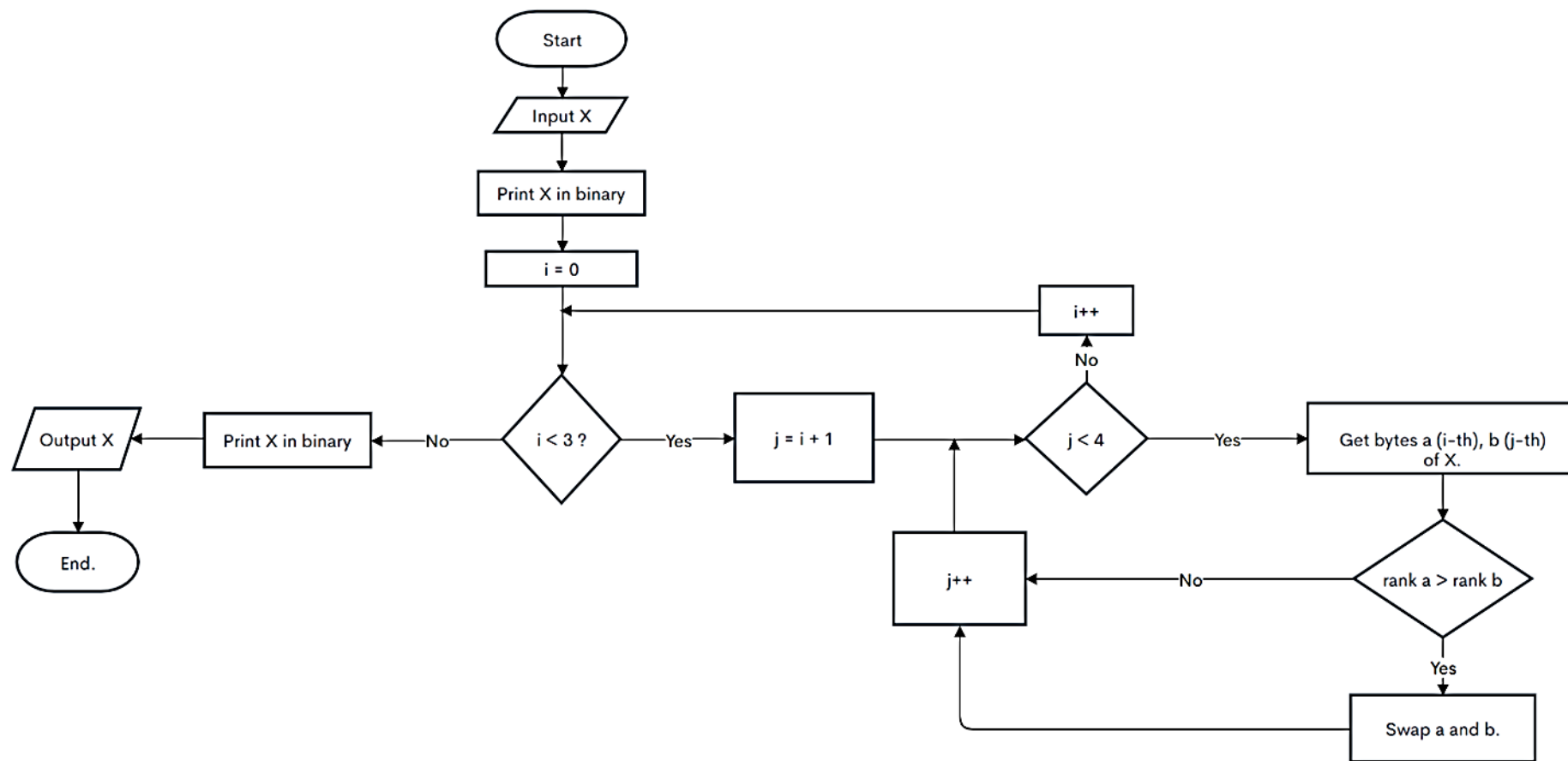
 Грозов В.А.

 **УНИВЕРСИТЕТ ИТМО**

Санкт-Петербург, 2020

Вариант 29. Назовем рангом байта результат побитовой импликации младшей и старшей тетрад. Упорядочить байты числа по возрастанию их рангов.

1. Блок-схема алгоритма преобразования для программы на С



## 2. Текст программы на С с комментариями.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void prtBin(int n){                                     //Выводить двоичное представление на
экран.
    int i = sizeof(int) * 8 - 1;
    while (!(n >> i & 1)){                             //Игнорировать бессмысленные нули
        i--;
    }
    for (i; i >= 0; i--){
        printf("%d", n >> i & 1);
        if (i%4 == 0){
            printf(" ");
        }
    }
    printf("\n");
}

int imp(int a, int b){                                 //Побитовая импликация.
    return (~a)|b;
}

int rankByte(int a){                                   //Ранг байта.
    int b = a & 15;
    a >>= 4;
    return imp(b,a);                                  //Импликация младшей и старшей тетрад.
}

int getByte(int x, int pos){ //Получить байт, который находится на позиции pos.
    return (x >> (8 * pos)) & 0x000000ff;
}

void putByte(int* x, int a, int pos){ //Поместить байт а в позицию pos
    *x = *x & (~(0xff << (pos * 8))); //pos-й байт = 0
    *x = *x | (a << (pos * 8));
}

int main()
{
    srand(time(NULL));                                //Получить случайное целое типа int.
    int x = rand();
    printf("x = %x\n", x);
    prtBin(x);                                         //Выводить его двоичное представление на экран.
    for (int i = 0; i<3; i++){//Упорядочить байты числа по возрастанию их рангов.
        for (int j = i + 1; j < 4; j++){
            int a = getByte(x, i);
            int b = getByte(x, j);
            if (rankByte(a) > rankByte(b)){           //Проверить условие.
                putByte(&x, a, j);                   //Поменять местами два байта.
                putByte(&x, b, i);
            }
        }
    }
    printf("x = %x\n", x);
    prtBin(x);                                         //Выводить на экран результат преобразования.
    return 0;                                          //Завершить программу.
}
```

### 3. Текст программы на ассемблере NASM с комментариями.

```
section .bss                                ;инициализирование памяти
    arr resb 64                             ;хранятся цифры числа
    arrPos resq 1                          ;адрес ячейки массива arr
    num resd 1                             ;хранится число
section .data
    i db 3
    a db 0
section .text
    global _start

_start:
    rdrand rax                            ;Получить случайное целое типа int.
    mov r12, 0xffffffffh
    and rax, r12
    mov [num], rax
    call _printBinary                    ;Выводить двоичное представление на экран.
    mov r8, [num]                       ;Упорядочить байты числа по возрастанию их рангов.
_bSort:
    mov r9, 0
    mov byte [a], 0

_comparison:
    mov rax, 8
    mul r9
    mov cl, al
    mov rax, 0xffff
    shl rax, cl
    and rax, r8                        ; rax - i-й байт, (i + 1)-й байт
    shr rax, cl
    mov bx, ax

    and rax, 0ffh                    ; rax - i-й байт
    xor rdx, rdx                    ;
    mov r10, 16
    div r10                        ;В dl хранит младшая тетрада, в al хранит старшая тетрада
    not dl                        ;Импликация
    or dl, al                    ;
    mov r11b, dl                  ;r11b - ганг i-о байта
    mov al, bh
    xor rdx, rdx                    ;
    div r10                        ;
    not dl                        ;
    or dl, al                    ; dl - ганг (i+1)-о байта
    cmp r11b, dl                  ;Проверить условие.
    jle _noswap                  ;Если ранг маленького байта больше чем, ранг большего байта
_swap:
    rol bx, 8                    ;то поменять местами два байта.
    and rbx, 0xffffh
    shl rbx, cl
```

```

    mov rax, 0ffffh
    shl rax, cl
    not rax
    and r8, rax
    or r8, rbx
    mov rax, r8
    mov byte[a], 1
_noswap:
    inc r9
    cmp r9b, [i]
    jnz _comparison
    cmp byte[a], 0           ;проверка конца цикла
    jnz _bSort

    mov rax, r8
    call _printBinary       ;Выводить на экран результат преобразования.
    mov rax, 60             ;Завершить программу.
    xor rdi, rdi
    syscall
_printBinary:
    mov rcx, arr
    mov rbx, 10             ; номер символа «перенос строки» в таблице ASCII
    mov [rcx], rbx
    inc rcx
    mov [arrPos], rcx
_pushDigit:
    mov rdx, 0
    mov rbx, 2              ;основание системы счисления
    div rbx
    push rax
    add rdx, 48             ;перевод символов в цифры
    mov rcx, [arrPos]
    mov [rcx], dl           ;Занесение цифр из регистра в массив
    inc rcx
    mov [arrPos], rcx
    pop rax
    cmp rax, 0
    jne _pushDigit
_printDigit:
    mov rcx, [arrPos]       ;извлечение цифр из массива
    mov rax, 1
    mov rdi, 1
    mov rsi, rcx
    mov rdx, 1
    syscall                 ;вывод на экран
    mov rcx, [arrPos]
    dec rcx
    mov [arrPos], rcx
    cmp rcx, arr            ;проверка конца цикла
    jge _printDigit
    ret

```

#### 4. Дизассемблерный листинг существенных частей программы на С с добавленными комментариями или пояснениями.

00000000000011e9 <prtBin>:

```

11e9:  f3 0f 1e fa      endbr64
11ed:  55               push    rbp
11ee:  48 89 e5         mov     rbp, rsp
11f1:  48 83 ec 20      sub     rsp, 0x20
11f5:  89 7d ec         mov     DWORD PTR [rbp-0x14], edi
11f8:  c7 45 fc 1f 00 00 00 mov     DWORD PTR [rbp-0x4], 0x1f
11ff:  eb 04           jmp     1205 <prtBin+0x1c>
1201:  83 6d fc 01      sub     DWORD PTR [rbp-0x4], 0x1
1205:  8b 45 fc         mov     eax, DWORD PTR [rbp-0x4]
1208:  8b 55 ec         mov     edx, DWORD PTR [rbp-0x14]
120b:  89 c1           mov     ecx, eax
120d:  d3 fa           sar     edx, cl
120f:  89 d0           mov     eax, edx
1211:  83 e0 01         and     eax, 0x1
1214:  85 c0           test    eax, eax
1216:  74 e9           je      1201 <prtBin+0x18>
1218:  eb 3a           jmp     1254 <prtBin+0x6b>
121a:  8b 45 fc         mov     eax, DWORD PTR [rbp-0x4]
121d:  8b 55 ec         mov     edx, DWORD PTR [rbp-0x14]
1220:  89 c1           mov     ecx, eax
1222:  d3 fa           sar     edx, cl
1224:  89 d0           mov     eax, edx
1226:  83 e0 01         and     eax, 0x1
1229:  89 c6           mov     esi, eax
122b:  48 8d 3d d2 0d 00 00 lea     rdi, [rip+0xdd2]      # 2004

```

<\_IO\_stdin\_used+0x4>

```

1232:  b8 00 00 00 00 00 mov     eax, 0x0
1237:  e8 84 fe ff ff   call    10c0 <printf@plt>
123c:  8b 45 fc         mov     eax, DWORD PTR [rbp-0x4]
123f:  83 e0 03         and     eax, 0x3
1242:  85 c0           test    eax, eax
1244:  75 0a           jne     1250 <prtBin+0x67>
1246:  bf 20 00 00 00 00 mov     edi, 0x20
124b:  e8 50 fe ff ff   call    10a0 <putchar@plt>
1250:  83 6d fc 01      sub     DWORD PTR [rbp-0x4], 0x1
1254:  83 7d fc 00      cmp     DWORD PTR [rbp-0x4], 0x0
1258:  79 c0           jns     121a <prtBin+0x31>
125a:  bf 0a 00 00 00 00 mov     edi, 0xa
125f:  e8 3c fe ff ff   call    10a0 <putchar@plt>
1264:  90             nop
1265:  c9             leave
1266:  c3             ret

```

0000000000001267 <imp>:

```

1267:  f3 0f 1e fa      endbr64
126b:  55               push    rbp
126c:  48 89 e5         mov     rbp, rsp
126f:  89 7d fc         mov     DWORD PTR [rbp-0x4], edi
1272:  89 75 f8         mov     DWORD PTR [rbp-0x8], esi
1275:  8b 45 fc         mov     eax, DWORD PTR [rbp-0x4]
1278:  f7 d0           not     eax
127a:  0b 45 f8         or      eax, DWORD PTR [rbp-0x8]
127d:  5d             pop     rbp
127e:  c3             ret

```

000000000000127f <rankByte>:

127f: f3 0f 1e fa  
1283: 55  
1284: 48 89 e5  
1287: 48 83 ec 18  
128b: 89 7d ec  
128e: 8b 45 ec  
1291: 83 e0 0f  
1294: 89 45 fc  
1297: c1 7d ec 04  
129b: 8b 55 ec  
129e: 8b 45 fc  
12a1: 89 d6  
12a3: 89 c7  
12a5: e8 bd ff ff ff  
12aa: c9  
12ab: c3

endbr64  
push rbp  
mov rbp, rsp  
sub rsp, 0x18  
mov DWORD PTR [rbp-0x14], edi  
mov eax, DWORD PTR [rbp-0x14]  
and eax, 0xf  
mov DWORD PTR [rbp-0x4], eax  
sar DWORD PTR [rbp-0x14], 0x4  
mov edx, DWORD PTR [rbp-0x14]  
mov eax, DWORD PTR [rbp-0x4]  
mov esi, edx  
mov edi, eax  
call 1267 <imp>  
leave  
ret

00000000000012ac <getByte>:

12ac: f3 0f 1e fa  
12b0: 55  
12b1: 48 89 e5  
12b4: 89 7d fc  
12b7: 89 75 f8  
12ba: 8b 45 f8  
12bd: c1 e0 03  
12c0: 8b 55 fc  
12c3: 89 c1  
12c5: d3 fa  
12c7: 89 d0  
12c9: 0f b6 c0  
12cc: 5d  
12cd: c3

endbr64  
push rbp  
mov rbp, rsp  
mov DWORD PTR [rbp-0x4], edi  
mov DWORD PTR [rbp-0x8], esi  
mov eax, DWORD PTR [rbp-0x8]  
shl eax, 0x3  
mov edx, DWORD PTR [rbp-0x4]  
mov ecx, eax  
sar edx, cl  
mov eax, edx  
movzx eax, al  
pop rbp  
ret

00000000000012ce <putByte>:

12ce: f3 0f 1e fa  
12d2: 55  
12d3: 48 89 e5  
12d6: 48 89 7d f8  
12da: 89 75 f4  
12dd: 89 55 f0  
12e0: 48 8b 45 f8  
12e4: 8b 00  
12e6: 8b 55 f0  
12e9: c1 e2 03  
12ec: be ff 00 00 00  
12f1: 89 d1  
12f3: d3 e6  
12f5: 89 f2  
12f7: f7 d2  
12f9: 21 c2  
12fb: 48 8b 45 f8  
12ff: 89 10  
1301: 48 8b 45 f8  
1305: 8b 00  
1307: 8b 55 f0  
130a: c1 e2 03  
130d: 8b 75 f4  
1310: 89 d1

endbr64  
push rbp  
mov rbp, rsp  
mov QWORD PTR [rbp-0x8], rdi  
mov DWORD PTR [rbp-0xc], esi  
mov DWORD PTR [rbp-0x10], edx  
mov rax, QWORD PTR [rbp-0x8]  
mov eax, DWORD PTR [rax]  
mov edx, DWORD PTR [rbp-0x10]  
shl edx, 0x3  
mov esi, 0xff  
mov ecx, edx  
shl esi, cl  
mov edx, esi  
not edx  
and edx, eax  
mov rax, QWORD PTR [rbp-0x8]  
mov DWORD PTR [rax], edx  
mov rax, QWORD PTR [rbp-0x8]  
mov eax, DWORD PTR [rax]  
mov edx, DWORD PTR [rbp-0x10]  
shl edx, 0x3  
mov esi, DWORD PTR [rbp-0xc]  
mov ecx, edx

1312:	d3 e6	shl	esi,cl
1314:	89 f2	mov	edx,esi
1316:	09 c2	or	edx,eax
1318:	48 8b 45 f8	mov	rax,QWORD PTR [rbp-0x8]
131c:	89 10	mov	DWORD PTR [rax],edx
131e:	90	nop	
131f:	5d	pop	rbp
1320:	c3	ret	

0000000000001321 <main>:

1321:	f3 0f 1e fa	endbr64	
1325:	55	push	rbp
1326:	48 89 e5	mov	rbp,esp
1329:	53	push	rbx
132a:	48 83 ec 28	sub	esp,0x28
132e:	64 48 8b 04 25 28 00	mov	rax,QWORD PTR fs:0x28
1335:	00 00		
1337:	48 89 45 e8	mov	QWORD PTR [rbp-0x18],rax
133b:	31 c0	xor	eax,eax
133d:	bf 00 00 00 00	mov	edi,0x0
1342:	e8 99 fd ff ff	call	10e0 <time@plt>
1347:	89 c7	mov	edi,eax
1349:	e8 82 fd ff ff	call	10d0 <srand@plt>
134e:	e8 9d fd ff ff	call	10f0 <rand@plt>
1353:	89 45 d4	mov	DWORD PTR [rbp-0x2c],eax
1356:	8b 45 d4	mov	eax,DWORD PTR [rbp-0x2c]
1359:	89 c6	mov	esi,eax
135b:	48 8d 3d a5 0c 00 00	lea	rdi,[rip+0xca5] # 2007
<_IO_stdin_used+0x7>			
1362:	b8 00 00 00 00	mov	eax,0x0
1367:	e8 54 fd ff ff	call	10c0 <printf@plt>
136c:	8b 45 d4	mov	eax,DWORD PTR [rbp-0x2c]
136f:	89 c7	mov	edi,eax
1371:	e8 73 fe ff ff	call	11e9 <prtBin>
1376:	c7 45 d8 00 00 00 00	mov	DWORD PTR [rbp-0x28],0x0
137d:	eb 7f	jmp	13fe <main+0xdd>
137f:	8b 45 d8	mov	eax,DWORD PTR [rbp-0x28]
1382:	83 c0 01	add	eax,0x1
1385:	89 45 dc	mov	DWORD PTR [rbp-0x24],eax
1388:	eb 6a	jmp	13f4 <main+0xd3>
138a:	8b 45 d4	mov	eax,DWORD PTR [rbp-0x2c]
138d:	8b 55 d8	mov	edx,DWORD PTR [rbp-0x28]
1390:	89 d6	mov	esi,edx
1392:	89 c7	mov	edi,eax
1394:	e8 13 ff ff ff	call	12ac <getBytes>
1399:	89 45 e0	mov	DWORD PTR [rbp-0x20],eax
139c:	8b 45 d4	mov	eax,DWORD PTR [rbp-0x2c]
139f:	8b 55 dc	mov	edx,DWORD PTR [rbp-0x24]
13a2:	89 d6	mov	esi,edx
13a4:	89 c7	mov	edi,eax
13a6:	e8 01 ff ff ff	call	12ac <getBytes>
13ab:	89 45 e4	mov	DWORD PTR [rbp-0x1c],eax
13ae:	8b 45 e0	mov	eax,DWORD PTR [rbp-0x20]
13b1:	89 c7	mov	edi,eax
13b3:	e8 c7 fe ff ff	call	127f <rankByte>
13b8:	89 c3	mov	ebx,eax
13ba:	8b 45 e4	mov	eax,DWORD PTR [rbp-0x1c]
13bd:	89 c7	mov	edi,eax
13bf:	e8 bb fe ff ff	call	127f <rankByte>



```

13c4: 39 c3      cmp     ebx,eax
13c6: 7e 28      jle     13f0 <main+0xcfc>
13c8: 8b 55 dc   mov     edx,DWORD PTR [rbp-0x24]
13cb: 8b 4d e0   mov     ecx,DWORD PTR [rbp-0x20]
13ce: 48 8d 45 d4 lea     rax,[rbp-0x2c]
13d2: 89 ce     mov     esi,ecx
13d4: 48 89 c7   mov     rdi,rax
13d7: e8 f2 fe ff ff call    12ce <putByte>
13dc: 8b 55 d8   mov     edx,DWORD PTR [rbp-0x28]
13df: 8b 4d e4   mov     ecx,DWORD PTR [rbp-0x1c]
13e2: 48 8d 45 d4 lea     rax,[rbp-0x2c]
13e6: 89 ce     mov     esi,ecx
13e8: 48 89 c7   mov     rdi,rax
13eb: e8 de fe ff ff call    12ce <putByte>
13f0: 83 45 dc 01 add     DWORD PTR [rbp-0x24],0x1
13f4: 83 7d dc 03 cmp     DWORD PTR [rbp-0x24],0x3
13f8: 7e 90      jle     138a <main+0x69>
13fa: 83 45 d8 01 add     DWORD PTR [rbp-0x28],0x1
13fe: 83 7d d8 02 cmp     DWORD PTR [rbp-0x28],0x2
1402: 0f 8e 77 ff ff ff jle     137f <main+0x5e>
1408: 8b 45 d4   mov     eax,DWORD PTR [rbp-0x2c]
140b: 89 c6     mov     esi,eax
140d: 48 8d 3d f3 0b 00 00 lea     rdi,[rip+0xbf3]          # 2007
<_IO_stdin_used+0x7>
1414: b8 00 00 00 00 mov     eax,0x0
1419: e8 a2 fc ff ff call    10c0 <printf@plt>
141e: 8b 45 d4   mov     eax,DWORD PTR [rbp-0x2c]
1421: 89 c7     mov     edi,eax
1423: e8 c1 fd ff ff call    11e9 <prtBin>
1428: b8 00 00 00 00 mov     eax,0x0
142d: 48 8b 5d e8 mov     rbx,QWORD PTR [rbp-0x18]
1431: 64 48 33 1c 25 28 00 xor     rbx,QWORD PTR fs:0x28
1438: 00 00
143a: 74 05     je      1441 <main+0x120>
143c: e8 6f fc ff ff call    10b0 <__stack_chk_fail@plt>
1441: 48 83 c4 28 add     rsp,0x28
1445: 5b       pop     rbx
1446: 5d       pop     rbp
1447: c3       ret
1448: 0f 1f 84 00 00 00 00 nop     DWORD PTR [rax+rax*1+0x0]
144f: 00

```

- Шестнадцатеричные числа слева, начиная с 0x11e9, являются адресами памяти.
- Второй столбец содержит инструкции машинного языка, которые процессор x64 считывает как двоичные значения. Например 01001110110111, objdump будет отображать двоичный файл как шестнадцатеричный, чтобы сделать его более удобочитаемым форматом.
- Последний правый столбец содержит ассемблерную версию инструкций машинного языка.

## 5. Краткий анализ по результатам сравнения программы на ассемблере и дизассемблированной программы на С.

- Дизассемблер - это компьютерная программа, которая переводит машинный язык на язык ассемблера - операция, обратная операции ассемблера.
- Дизассемблированная программа на С длинее, чем программа на ассемблере.
- Программа, написанная на языке ассемблера, может состоять из нескольких частей, называемых модулями. В каждом модуле могут быть определены один или несколько сегментов данных, стека и кода. Любая законченная программа на ассемблере должна включать один главный, или основной, модуль, с которого начинается ее выполнение.

## 6. Скриншоты прогонов программ на различных исходных данных

```
hanhnguyen26@ubuntu:~/Documents$ gcc -o lab2 lab2.c
hanhnguyen26@ubuntu:~/Documents$ ./lab2
x = c9458b2
1100 1001 0100 0101 1000 1011 0010
x = b294580c
1011 0010 1001 0100 0101 1000 0000 1100
hanhnguyen26@ubuntu:~/Documents$ nasm -f elf64 lab02.asm
hanhnguyen26@ubuntu:~/Documents$ ld -o lab02 lab02.o
hanhnguyen26@ubuntu:~/Documents$ ./lab02
1100100101000101100010110010
10110010100101000101100000001100
hanhnguyen26@ubuntu:~/Documents$ gcc -o lab2 lab2.c
hanhnguyen26@ubuntu:~/Documents$ ./lab2
x = 5459693c
101 0100 0101 1001 0110 1001 0011 1100
x = 5459693c
101 0100 0101 1001 0110 1001 0011 1100
hanhnguyen26@ubuntu:~/Documents$ nasm -f elf64 lab02.asm
hanhnguyen26@ubuntu:~/Documents$ ld -o lab02 lab02.o
hanhnguyen26@ubuntu:~/Documents$ ./lab02
1010100010110010110100100111100
1010100010110010110100100111100
hanhnguyen26@ubuntu:~/Documents$ gcc -o lab2 lab2.c
hanhnguyen26@ubuntu:~/Documents$ ./lab2
x = 36911767
11 0110 1001 0001 0001 0111 0110 0111
x = 91673617
1001 0001 0110 0111 0011 0110 0001 0111
hanhnguyen26@ubuntu:~/Documents$ nasm -f elf64 lab02.asm
hanhnguyen26@ubuntu:~/Documents$ ld -o lab02 lab02.o
hanhnguyen26@ubuntu:~/Documents$ ./lab02
110110100100010001011101100111
10010001011001110011011000010111
```