ОТЧЕТ
О ВЫПЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ № 2

Задание.

№Варианта	Название алгоритма	Режим шифрования
10	ГОСТ 28147-89	OFB

Общие сведения.

Алгоритм шифрования ГОСТ 28147-89 является симметричным, блочным алгоритмом. Преобразование осуществляется над блоком размером 64 бита, размер секретного ключа 256 бит, в алгоритме 32 раунда преобразований.

Необходимые определения и обозначения:

X – блок открытого текста размером 64 бита;

Y – блок зашифрованного текста размером 64 бита;

K – секретный ключ (256 бит);

W – раундовый ключ.

В алгоритме ГОСТ используются следующие операции:

S-блок или S-box – табличная подстановка, при которой группа бит отображается в другую группу бит;

$\boxplus$ – операция сложения по модулю 2^{32};

$\oplus$ или XOR – операция сложения по модулю 2 (или побитовое «исключающее или»);

$\hookleftarrow 11$ – циклический сдвиг влево на 11 бит.

Эти операции циклически повторяются в алгоритме, образуя так называемые раунды. Входом каждого раунда является выход предыдущего раунда и раундовый подключ W_i, который получен из секретного ключа шифрования K следующим образом. Рассмотрим секретный ключ K (256 бит), состоящий из восьми слов по 32 бита: $K = K_0 K_1 K_2 K_3 K_4 K_5 K_6 K_7$. На их основе строим раундовый ключ: W Основная схема алгоритма ГОСТ представлена на рисунке 1

$$W = \underbrace{K_0 K_1 K_2 K_3 K_4 K_5 K_6 K_7}_{\text{Раунды 1-8}} \underbrace{K_0 K_1 K_2 K_3 K_4 K_5 K_6 K_7}_{\text{Раунды 9-16}} \underbrace{K_0 K_1 K_2 K_3 K_4 K_5 K_6 K_7}_{\text{Раунды 17-24}} \underbrace{K_7 K_6 K_5 K_4 K_3 K_2 K_1 K_0}_{\text{Раунды 25-32}}$$

.

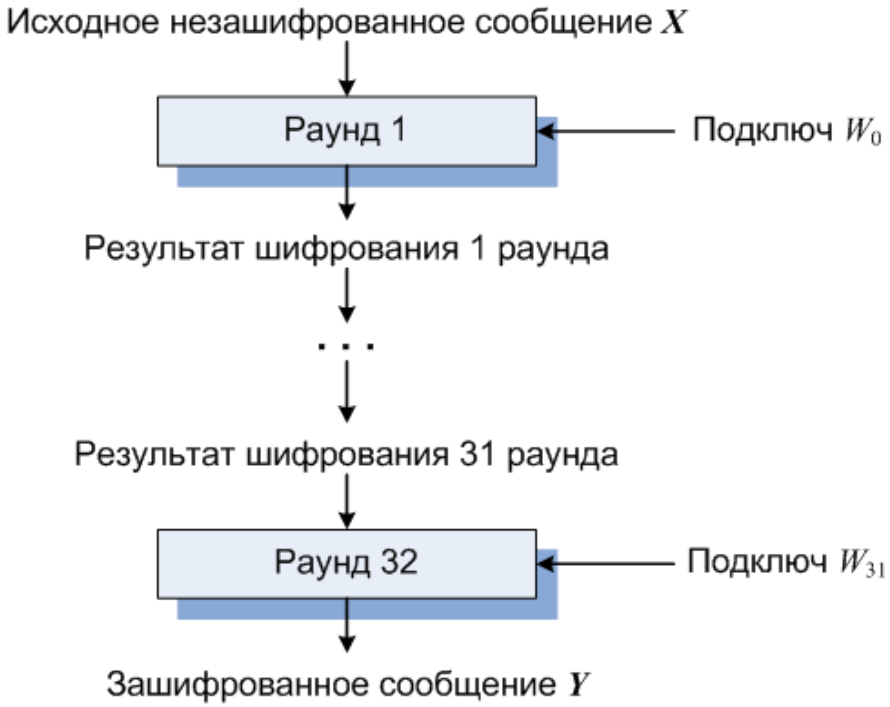


Рис. 1. Основная схема алгоритма ГОСТ

Для шифрования блок открытого текста сначала разбивается на две одинаковые части, правую *R* (младшее слово) и левую *L* (старшее слово) рисунок 2.

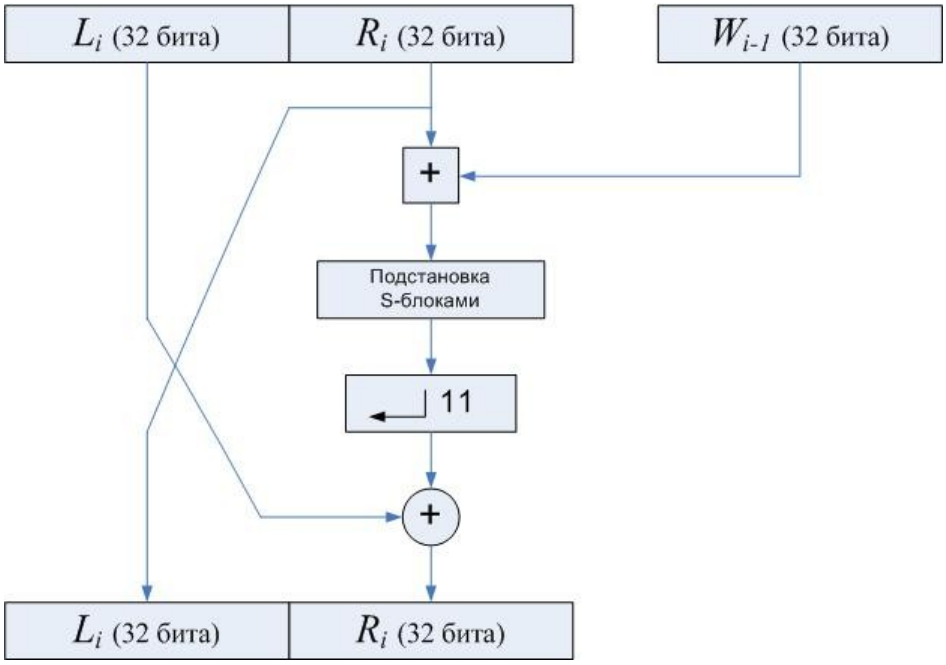


Рис. 2. Схема одного раунда алгоритма ГОСТ

На *i*-м раунде используется подключ W_{i-1}. Правая часть R_i складывается по модулю 2^{32} с раундовым подключом W_{i-1}. Над получившимся результатом выполняется операция табличной подстановки рисунок 3.

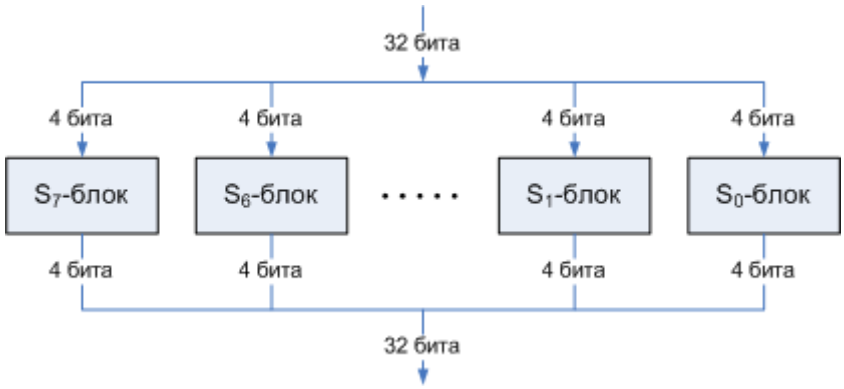


Рис. 3. Подстановка S-блоками

Для этого результат разбивается на восемь 4-битовых кусочков, каждый из которых подается на вход своего S-блока: первые четыре бита в S_0-блок, вторые – в S_1-блок и так далее. Каждый S-блок содержит 16 четырехбитовых элементов, нумеруемых с 0 по 15. ГОСТ рекомендует заполнять каждую из восьми таблиц различными числами множества {0, 1, 2, …, 15}, переставленными случайным образом.

S_0-блок

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Элемент	4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3

S_1-блок

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Элемент	14	11	4	12	6	13	15	10	2	3	8	1	0	7	5	9

S_2-блок

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Элемент	5	8	1	13	10	3	4	2	14	15	12	7	6	0	9	11

S_3-блок

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Элемент	7	13	10	1	0	8	9	15	14	4	6	12	11	2	5	3

S_4-блок

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Элемент	6	12	7	1	5	15	13	8	4	10	9	14	0	3	11	2

S_5-блок

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Элемент	4	11	10	0	7	2	1	13	3	6	8	5	9	12	15	14

S_6-блок

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Элемент	13	11	4	1	3	15	5	9	0	10	14	7	6	8	2	12

S_7-блок

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Элемент	1	15	13	0	5	7	10	4	9	2	3	14	6	11	8	12

Рис. 4. Набор S-блоков Центрального банка РФ

По входным четырем битам определяется номер элемента в S-блоке, который поступает на выход. Выходы всех восьми S-блоков объединяются в 32-битовое слово, затем все слово циклически сдвигается влево на 11 бит. Наконец, результат объединяется с помощью XOR с левой половиной, и получается новая правая половина, а правая половина становится левой половиной. Эти операции выполняются 32 раза. После этого левая и правая части меняются местами.

Запишем базовый цикл алгоритма ГОСТ.

Вход: Блок L, R, раундовый ключ W.

Выход: Преобразованный блок L, R.

FOR $i = 0$ TO 31 DO

$k \leftarrow R \boxplus W_i;$

$k = (k_7 \dots k_0)_{16};$

FOR $j = 0$ TO 7 DO $k_j \leftarrow S_j[k_j];$

$L \leftarrow L \oplus (k \hookleftarrow 11);$

$L \leftrightarrow R;$

$L \leftrightarrow R;$

RETURN $L, R.$

Для шифрования и дешифрования сообщения используется один алгоритм. Единственным различием является генерация раундового ключа. Чтобы дешифровать блок, строим раундовый ключ

$$W = \underbrace{K_0 K_1 K_2 K_3 K_4 K_5 K_6 K_7}_{\text{Раунды 1-8}} \underbrace{K_7 K_6 K_5 K_4 K_3 K_2 K_1 K_0}_{\text{Раунды 9-16}} \underbrace{K_7 K_6 K_5 K_4 K_3 K_2 K_1 K_0}_{\text{Раунды 17-24}} \underbrace{K_7 K_6 K_5 K_4 K_3 K_2 K_1 K_0}_{\text{Раунды 25-32}},$$

подаем на вход Y и на выходе получаем X.

ЛИСТИНГ ПРОГРАММЫ

```
#include "stdafx.h"
#include <iostream>

#define OPENFILE        "D:\\opentext.txt"
#define CRYPTEDFILE     "D:\\cryptedtext.txt"
#define DECRYPTEDFILE   "D:\\decryptedtext.txt"

#define SEKRETKEY       "98765432109876543210987654321098"

#define CRYPT           false
#define DECRYPT         true

using namespace std;

int reverse_bit(int X)
{
   return
   X =                          // Реверсирование битов
        ((X & 0x1)          <<31)+((X & 0x2)        <<29)+((X & 0x4)
<<27)+((X & 0x8)       <<25)+
        ((X & 0x10)         <<23)+((X & 0x20)       <<21)+((X & 0x40)
<<19)+((X & 0x80)      <<17)+
        ((X & 0x100)        <<15)+((X & 0x200)      <<13)+((X & 0x400)
<<11)+((X & 0x800)     << 9)+
        ((X & 0x1000)       << 7)+((X & 0x2000)     << 5)+((X & 0x4000)
<< 3)+((X & 0x8000)    << 1)+
        ((X & 0x10000)      >> 1)+((X & 0x20000)    >> 3)+((X & 0x40000)
>> 5)+((X & 0x80000)    >> 7)+
       ((X & 0x100000)     >> 9)+((X & 0x200000)   >>11)+((X & 0x400000)
>>13)+((X & 0x800000)   >>15)+
          ((X & 0x1000000)     >>17)+((X & 0x2000000)  >>19)+((X &
0x4000000)  >>21)+((X & 0x8000000)  >>23)+
          ((X & 0x10000000)    >>25)+((X & 0x20000000) >>27)+((X &
0x40000000) >>29)+((X & 0x80000000) >>31);
}

int reverse_block(int X)
```

```
{
   return
    X =                          // Реверсирование 8-битных блоков + реверс
битов результата
       ((X & 0x1)       << 7)+((X & 0x2)       << 5)+((X & 0x4)       << 3)+
((X & 0x8)       << 1)+
       ((X & 0x10)       >> 1)+((X & 0x20)       >> 3)+((X & 0x40)       >>
5)+((X & 0x80)       >> 7)+
       ((X & 0x100)       << 7)+((X & 0x200)       << 5)+((X & 0x400)       <<
3)+((X & 0x800)       << 1)+
       ((X & 0x1000)       >> 1)+((X & 0x2000)       >> 3)+((X & 0x4000)
>> 5)+((X & 0x8000)       >> 7)+
       ((X & 0x10000)       << 7)+((X & 0x20000)       << 5)+((X & 0x40000)
<< 3)+((X & 0x80000)       << 1)+
       ((X & 0x100000)       >> 1)+((X & 0x200000)       >> 3)+((X & 0x400000)
>> 5)+((X & 0x800000)       >> 7)+
          ((X & 0x1000000)       << 7)+((X & 0x2000000)       << 5)+((X &
0x4000000)       << 3)+((X & 0x8000000)       << 1)+
          ((X & 0x10000000)       >> 1)+((X & 0x20000000)       >> 3)+((X &
0x40000000)       >> 5)+((X & 0x80000000)       >> 7);
}

void GOST(bool mode)
{
   //--8 S-box'ов----------------------------------------
   unsigned int Sbox[8][16] =
   {
      {4,10,9,2,13,8,0,14,6,11,1,12,7,15,5,3},
      {14,11,4,12,6,13,15,10,2,3,8,1,0,7,5,9},
      {5,8,1,13,10,3,4,2,14,15,12,7,6,0,9,11},
      {7,13,10,1,0,8,9,15,14,4,6,12,11,2,5,3},
      {6,12,7,1,5,15,13,8,4,10,9,14,0,3,11,2},
      {4,11,10,0,7,2,1,13,3,6,8,5,9,12,15,14},
      {13,11,4,1,3,15,5,9,0,10,14,7,6,8,2,12},
      {1,15,13,0,5,7,10,4,9,2,3,14,6,11,8,12}
   };

   //--Ключ шифрования и его разбиение-------------------
   char s[33] = SEKRETKEY;
```

```
unsigned int key[8];

for(int i = 0; i<8; i++)
{
    key[i] = (s[4*i]<<24) + (s[1+4*i]<<16) + (s[2+4*i]<<8) + s[3+4*i];  // Складывание каждых 4-х символов ключа в одно число

   key[i] = reverse_bit(key[i]);       // Реверсирование битов
}

//--Открытие файла-----------------------------------
FILE* f;           // Входной файл
FILE* g;           // Выходной файл
int fsize;         // Размер входного файла

if(mode == CRYPT)
{
   f = fopen(OPENFILE, "r");

   fseek (f, 0, SEEK_END);        // Вычисление размера файла
   fsize = ftell(f);
   fseek (f, 0, SEEK_SET);

   g = fopen(CRYPTEDFILE, "w");
}
if(mode == DECRYPT)
{
   f = fopen(CRYPTEDFILE, "r");

   fseek (f, 0, SEEK_END);        // Вычисление размера файла
   fsize = ftell(f);
   fseek (f, 0, SEEK_SET);

   g = fopen(DECRYPTEDFILE, "w");
}

//--Основной цикл шифрования-------------------------
while(fsize)
{
```

```
unsigned int A = 0;              // Страший 32-битный блок
unsigned int B = 0;              // Младший 32-битный блок

//-----------------------------------
if(fsize>=4)                 // Заполнение старшего блока...
{
   fread(&A, 4, 1, f);       // ...символами...
   fsize -= 4;
}
else
{
   fread(&A, fsize, 1, f);
   for(int i = 0; i<(4-fsize); i++)
   {
        A += (32<<(24-(i*8)));  // ... или пробелами, если символы кончились,
      fsize = 0;            //     а 64 бита еще не набралось
   }
}

//-----------------------------------
if(fsize>=4)                 // Заполнение младшего блока...
{
   fread(&B, 4, 1, f);       // ...символами...
   fsize -= 4;
}
else
{
   fread(&B, fsize, 1, f);
   for(int i = 0; i<(4-fsize); i++)
   {
        B += (32<<(24-(i*8)));  // ... или пробелами, если символы кончились,
      fsize = 0;            //     а 64 бита еще не набралось
   }
}
//-----------------------------------
    A = reverse_block(A);            // Реверсирование блоков с последующим реверсом бит
```

```
            B = reverse_block(B);              // Реверсирование блоков с последующим реверсом бит

            unsigned int T;              // Для промежуточных вычислений

            //--32 раунда-----------------------------------
            for(int i = 0; i<32; i++)
            {
               T = 0;
               if(mode == CRYPT)
               {
                  if(i<24) T = (A+key[i%8]) % 0x100000000;    // суммирование с ключом в зависимости от раунда
                  else T = (A+key[7-(i%8)]) % 0x100000000;
               }
               if(mode == DECRYPT)
               {
                  if(i<8) T = (A+key[i%8])  % 0x100000000;    // суммирование с ключом в зависимости от раунда
                  else T = (A+key[7-(i%8)]) % 0x100000000;
               }

                  unsigned int Fragments[8] =      // Разбиение на 4-битные фрагменты
               {
                  (T & 0xF0000000)>>28,
                  (T & 0xF000000)>>24,
                  (T & 0xF00000)>>20,
                  (T & 0xF0000)>>16,
                  (T & 0xF000)>>12,
                  (T & 0xF00)>> 8,
                  (T & 0xF0)>> 4,
                  (T & 0xF)
               };

               for(int j = 0; j<8; j++)
               {
                        Fragments[j] = Sbox[j][Fragments[j]-1];       // Пропуск фрагментов через Sbox'ы
```

```
            }

            T = (Fragments[0]<<28) +        // Сборка фрагментов обратно в 32-битный подблок
               (Fragments[1]<<24) +
               (Fragments[2]<<20) +
               (Fragments[3]<<16) +
               (Fragments[4]<<12) +
               (Fragments[5]<< 8) +
               (Fragments[6]<< 4) +
                Fragments[7];

             T = (T<<11)|(T>>21);           // Циклическое смещение влево на 11 бит
            T ^= B;                     // XOR с B

            if(i != 31)
            {
                B = A;                  // Перестановка подблоков 1-31 и 32 циклов
               A = T;
            }
            else
            {
               B = T;
            }
          }  //--Коненц 32 раундов--------------------------

             A = reverse_block(A);          // Реверсирование блоков с последующим реверсом бит
             B = reverse_block(B);          // Реверсирование блоков с последующим реверсом бит

          //--Вывод A и B в файл-------------------------
          fwrite(&A, 4, 1, g);
          fwrite(&B, 4, 1, g);

        }  // Конец файла
        fclose(f);
```

```
    fclose(g);
}


int _tmain(int argc, _TCHAR* argv[])
{
    GOST(CRYPT);
    GOST(DECRYPT);

    return 0;
}
                for(int j=0;j<8;j++)        //Заполняем блок битами символов
(число символов N=8)
                {
                    if(n/Pow(2,7-j)>=1)
                    {
                        T[8*i+j]=1;
                        n=n-Pow(2,7-j);
                    }
                    else
                        T[8*i+j]=0;
                }
            }
        }
    //Вывод блока T
        fprintf(log,"Tnach ");
        for(int i=0;i<64;i++)
            fprintf(log,"%d",T[i]);
}

void ZapolnB()                  //Заполнение блока B
{
    //Заполнение и вывод блока B1
        fprintf(log,"\nB1    ");
        for(int i=0;i<32;i++)
        {
            B[i]=T[i];
            fprintf(log,"%d",B[i]);
        }
```

```
}

void ZapolnA()                    //Заполнение блока A
{
  //Заполнение и вывод блока A1
    fprintf(log,"\nA1    ");
    for(int i=0;i<32;i++)
    {
      A[i]=T[32+i];
      fprintf(log,"%d",A[i]);
    }
}


void ZapolnK(int *t)              //Заполнение и вывод подключа K
{
  fprintf(log,"\nK%-5d",*t);
  unsigned long x=0;
  if (rejim==1)                   //Если режим шифрования, то
   {                                    //последовательность подключей имеет
следующий вид:
    if((*t>=25)&&(*t<=32))
      x=KEY[7-(*t-1)%8];
    else
      x=KEY[(*t-1)%8];
  }
  if (rejim==2)                   //Если режим шифрования, то
   {                                    //последовательность подключей имеет
следующий вид:
    if((*t>=1)&&(*t<=8))
      x=KEY[(*t-1)%8];
    else
      x=KEY[7-(*t-1)%8];
  }
   for(int i=0;i<32;i++)          //Переводим значение подключа в массив
битов
  {
    if(x/Pow(2,31-i)>=1)
    {
```

```
            K[i]=1;
            x=x-Pow(2,31-i);
          }
          else
            K[i]=0;
         fprintf(log,"%d",K[i]);       //Вывод подключа
       }
     }

     void Sum232()                    //Заполнение блока K
     {
       fprintf(log,"\nSUM32 ");
       for(int c=0,i=31;i>=0;i--)      //Поразрядное сложение блоков
       {
         if ((A[i]+K[i]+c)>=2)       //Если переполнение, то:
         {
           SUM32[i]=A[i]+K[i]+c-2;
           c=1;
         }
         else                        //Иначе:
         {
           SUM32[i]=A[i]+K[i]+c;
           c=0;
         }
       }
       for(int i=0;i<32;i++) //вывод сумматора
         fprintf(log,"%d",SUM32[i]);
     }

     void ZapolnN()                   //Заполнение накопителя
     {
       fprintf(log,"\nN     ");
       for(int i=0;i<8;i++)           //В накопителе 8 чисел
       {
          for(int j=0;j<4;j++)        //Преобразуем 4 бита сумматора (начиная с
конца, т.е. 4 посл бита, 4 предпосл бита и т.д.) в 10-чное число
           if(SUM32[31-4*i-j]==1)
                N[i]=N[i]+Pow(2,j);
         fprintf(log,"%d ",N[i]);    //Выводим накопитель
```

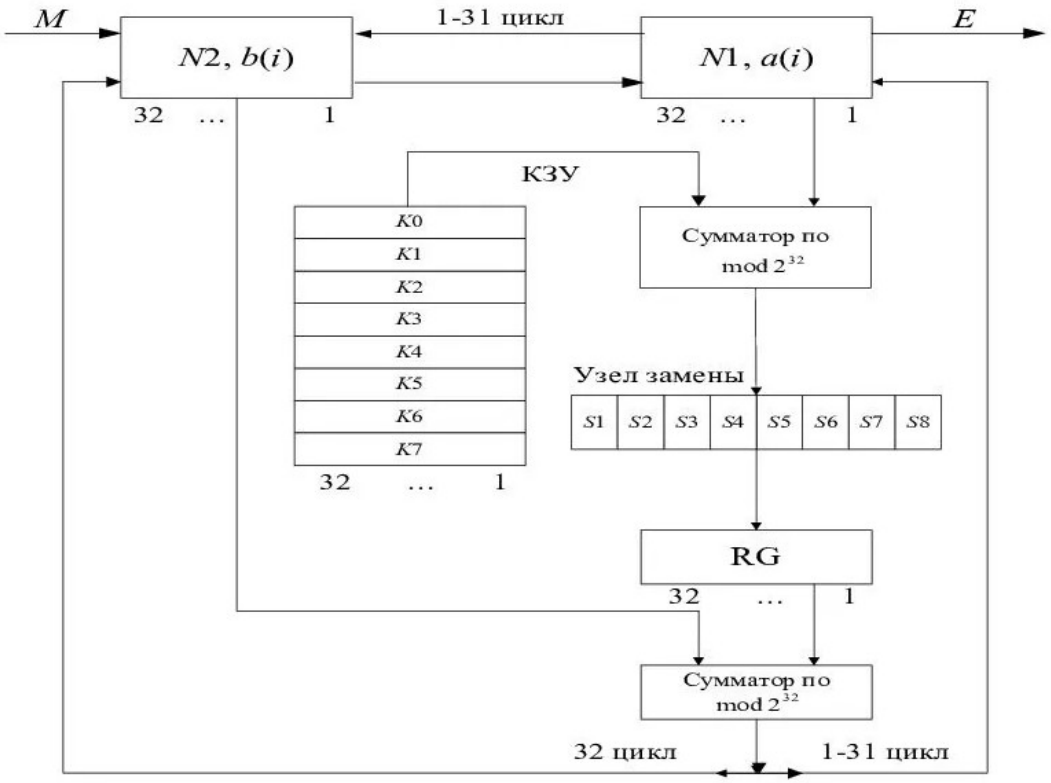
```
    }
  }

  void UzliZamen()                    //Прохождение заполнителя через узлы замен
  {
    fprintf(log,"%\nN     ");
    for(int i=0,k;i<8;i++)
    {
      k=N[i];
      N[i]=TABLE[i][k];          //Выход из i-ого узла
      fprintf(log,"%d ",N[i]);
    }
  }

  void ZapolnN1()                     //Преобразование результатов функции UzliZamen() в накопитель N1
  {
    for(int i=0;i<8;i++)          //Все 8 чисел накопителя N
      for(int j=0;j<4;j++)        //переводим в двоичный вид
      {
        if(N[i]/Pow(2,3-j)>=1)
        {
          N1[4*i+j]=1;
          N[i]=N[i]-Pow(2,3-j);
        }
        else
          N1[4*i+j]=0;
      }
    fprintf(log,"%\nN1    ");
    for(int i=0;i<32;i++) //вывод выходного накопителя
      fprintf(log,"%d",N1[i]);
  }

  void Sdvig11()                      //Сдвиг на 11 бит влево. Результат сдвига - результа функции Фейстеля F(Ai,Ki)
  {
    fprintf(log,"\nF     ");
    for(int i=0;i<21;i++)
```

```
        {
           F[i]=N1[i+11];          //Сохраняем биты с 12 по 32
           fprintf(log,"%d",F[i]);
        }
        for(int i=0;i<11;i++)
        {
           F[i+21]=N1[i];          //Сохраняем биты с 1 по 11
           fprintf(log,"%d",F[i+21]);
        }
}

void XOR(int *t)                   //Ai+1 = Bi XOR F(Ai,Ki)
{
   fprintf(log,"\nA%-5d",*t+1);
   for(int i=0;i<32;i++)
   {
      A[i]=F[i]^B[i];              //Двоичное исключающее "ИЛИ"
      fprintf(log,"%d",A[i]);
   }
}

void SohrBnext(int *t)             //Bi+1 = Ai
{
   fprintf(log,"\nB%-5d",*t+1);
   for(int i=0;i<32;i++)
   {
      B[i]=C[i];
      fprintf(log,"%d",B[i]);
   }
}

void SkleivanieAB()                //Склеивание блоков. T=AB
{
   fprintf(log,"\nTkon  ");
   for(int i=0;i<32;i++)
      T[i]=A[i];
   for(int i=32;i<64;i++)
      T[i]=B[i-32];
   for(int i=0;i<64;i++)
```

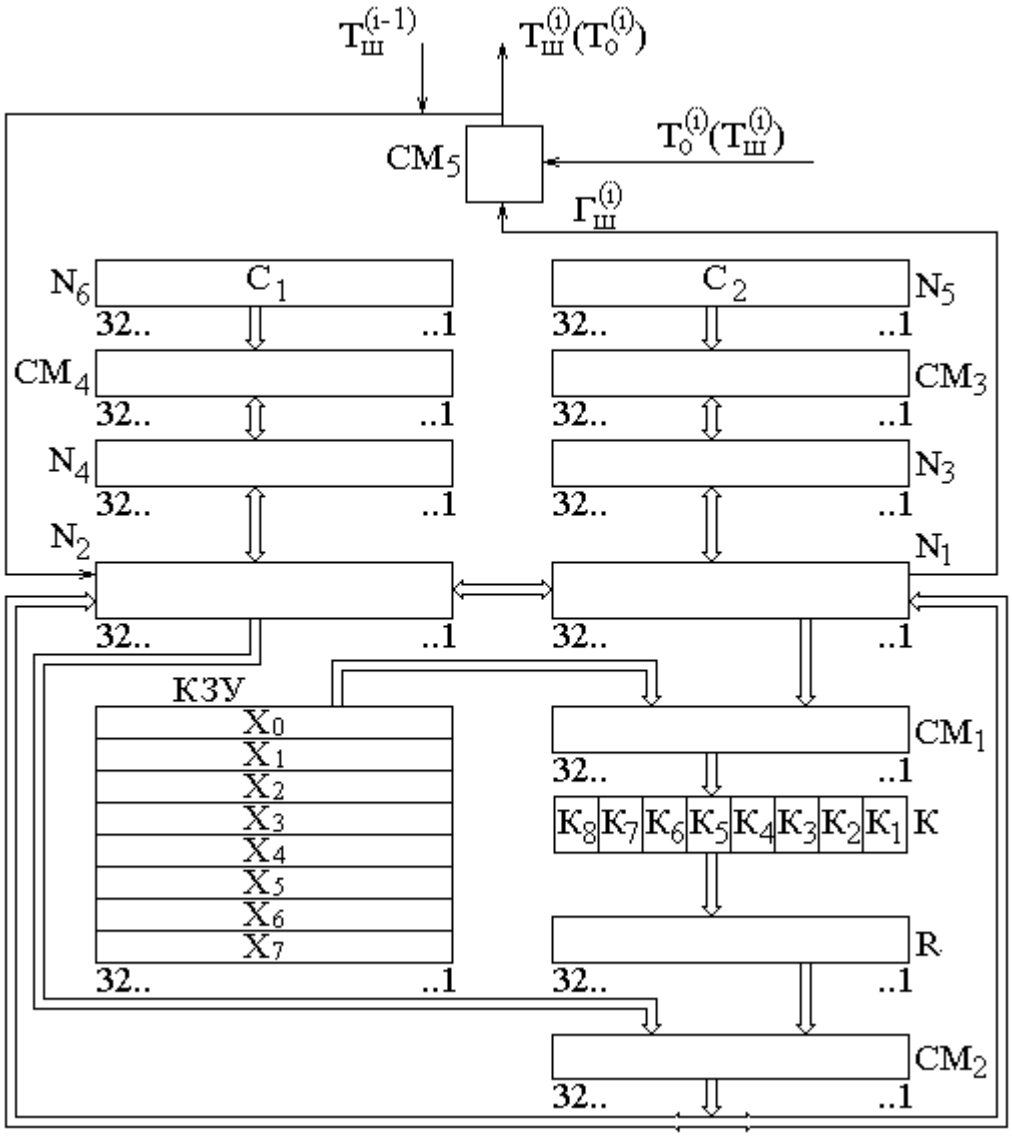
```
            fprintf(log,"%d",T[i]);
        }

        void ViviodTexta(int *x)                //Преобразование блока T в символы и вывод этих символов в выходной файл (fo)
        {
           int d;
           fprintf(log,"\n\nRezultat: ");
            if((*x==blokN-1)&&(nekratno==1)&&(rejim==2))     //Если последний блок, "..." - некратно и режим расшифрования, то
               d=FileSize(fi)%8;                    //выводим первые d=FileSize(fi)%8 символов.
           else                                 //Иначе
              d=8;                               //выводим 8 символов
           for(int i=0;i<d;i++)
           {
              n=0;
                for(int j=0;j<8;j++)                    //Преобразование из 8 бит в соответствующий символ
                  if (T[8*i+j]==1)
                     n=n+Pow(2,7-j);
              ch=n;
              fprintf(log,"%c",ch);
              fprintf(fo,"%c",ch);
           }
           fprintf(log,"\n");
        }
```

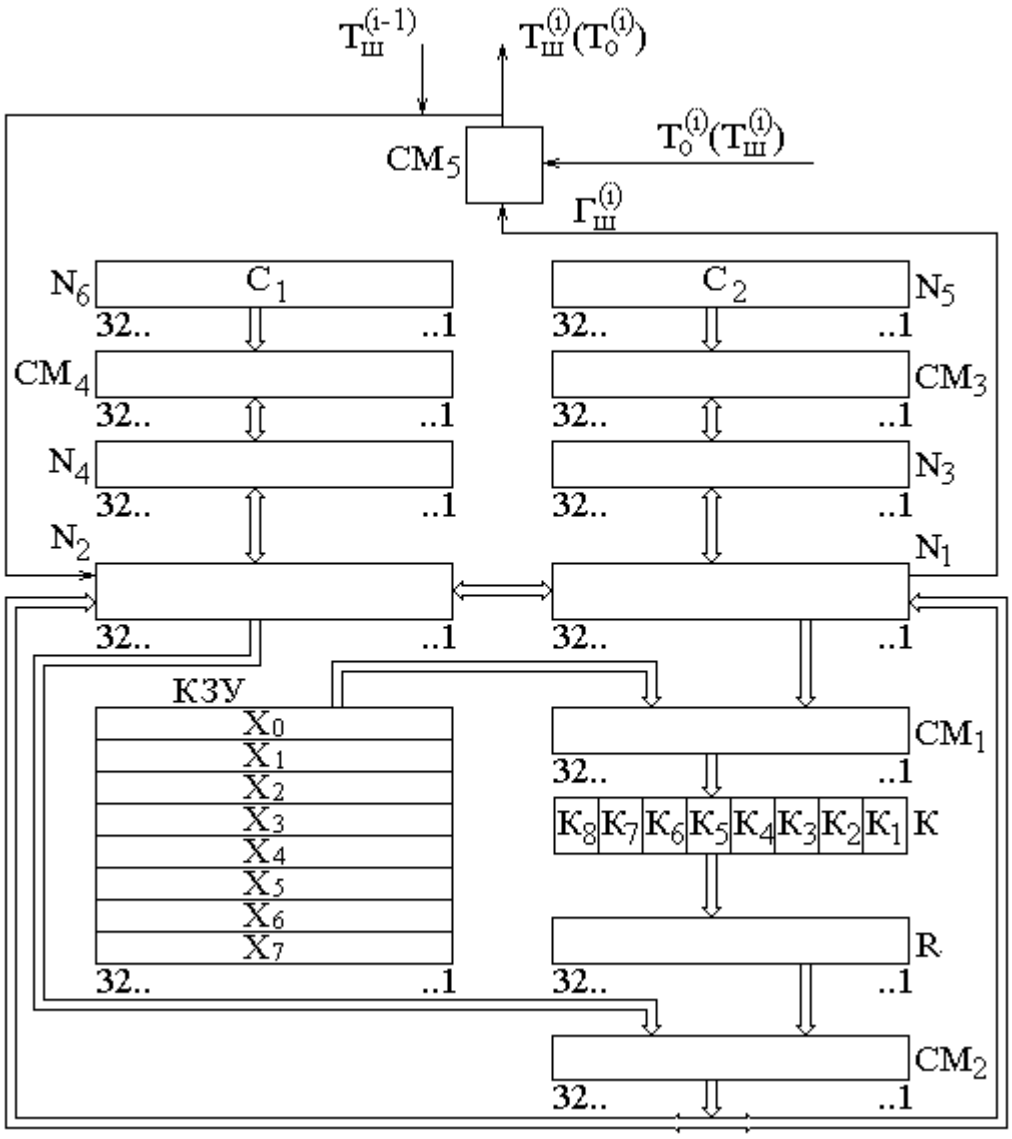
ПРИЛОЖЕНИЕ

Шифрование в режиме простой замены



Структурная схема алгоритма




$T_{ш}^{(i-1)}$
$T_{ш}^{(i)}(T_{о}^{(i)})$
$T_{о}^{(i)}(T_{ш}^{(i)})$
$СМ_5$
$Г_{ш}^{(i)}$
N_6
C_1
C_2
N_5
32..
..1
$СМ_4$
$СМ_3$
N_4
N_3
N_2
N_1
КЗУ
X_0
X_1
X_2
X_3
X_4
X_5
X_6
X_7
$СМ_1$
K_8 K_7 K_6 K_5 K_4 K_3 K_2 K_1 K
R
$СМ_2$

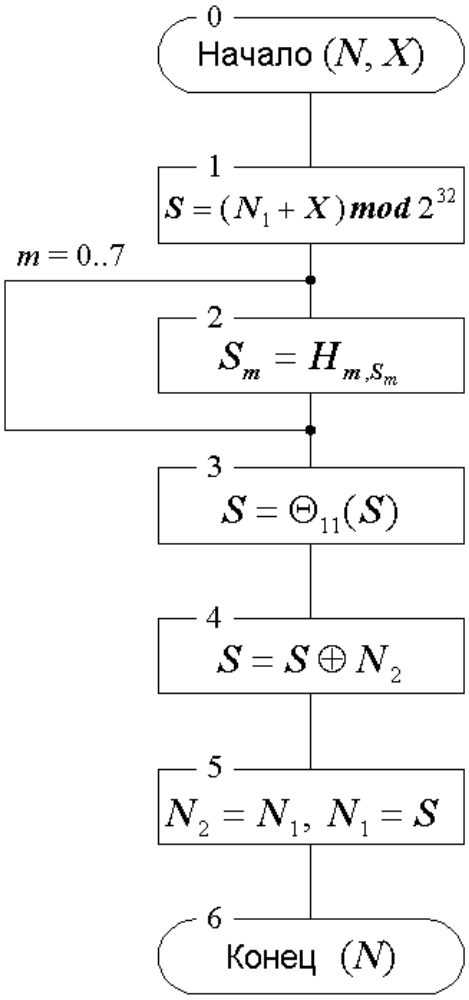


Схема основного шага криптопреобразования алгоритма ГОСТ 28147-89.

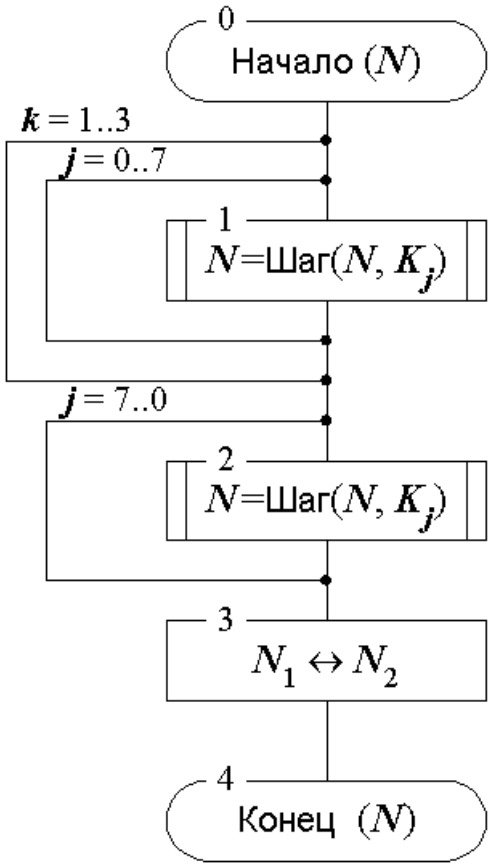


Схема цикла шифрования 32-З.

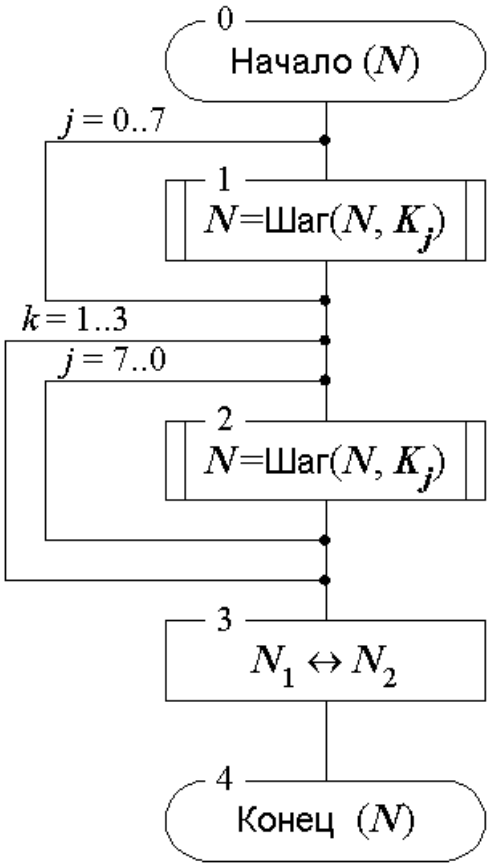
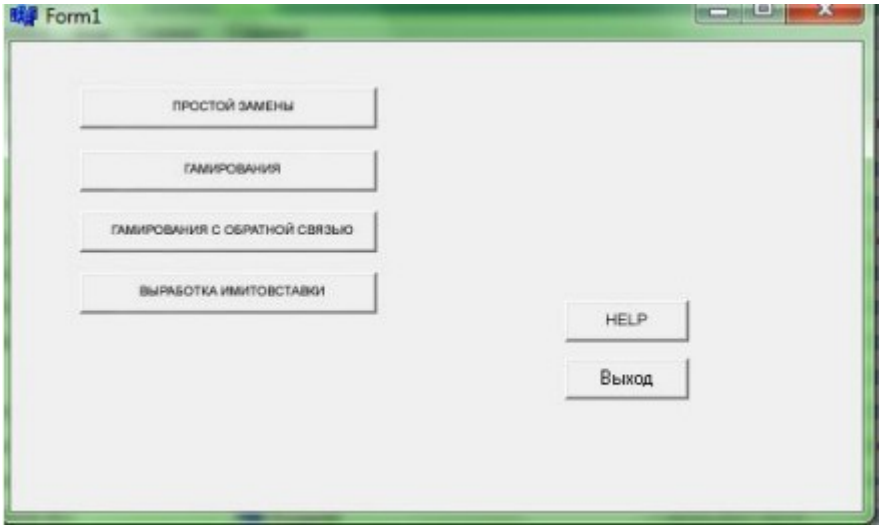
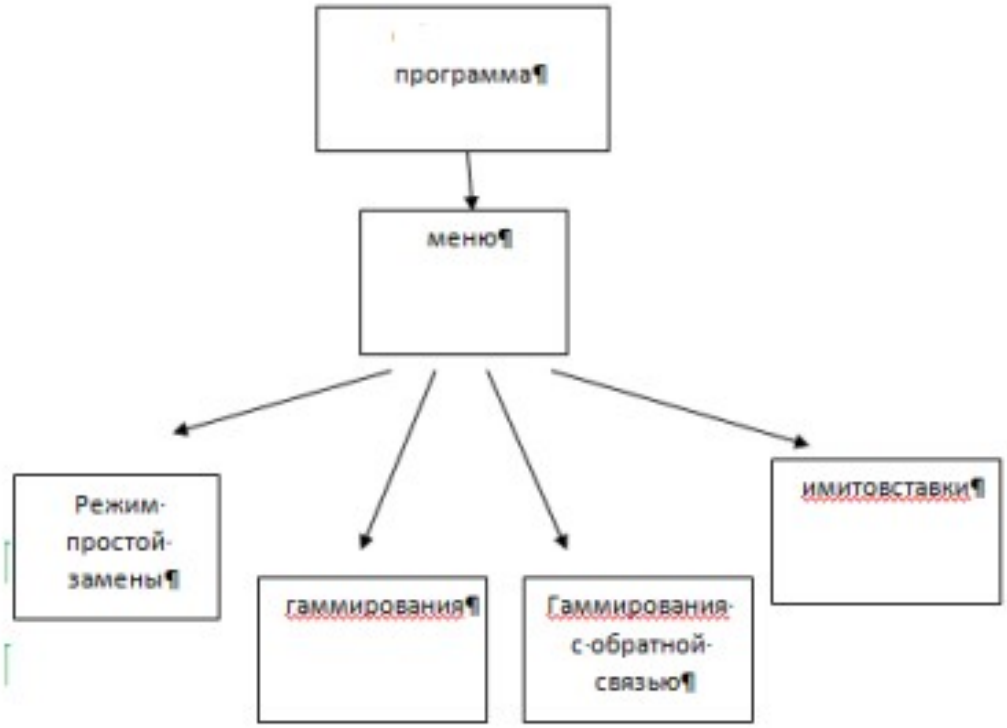


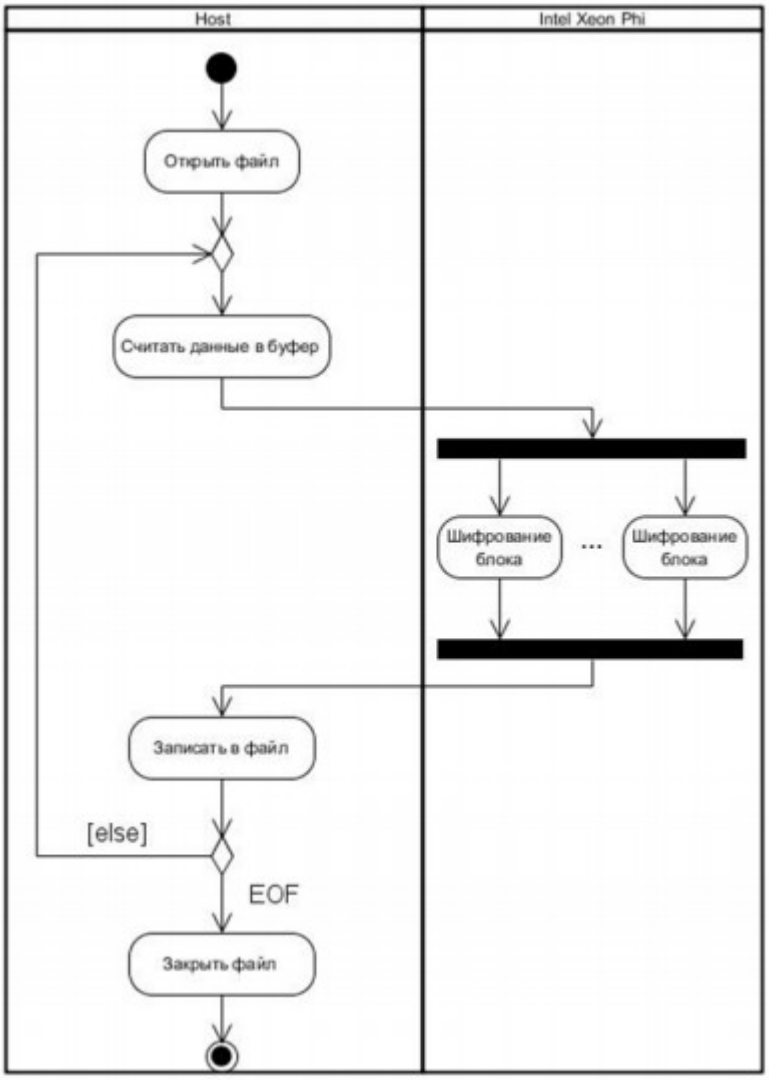
Схема цикла дешифрования 32-Р.

Главное окно программы

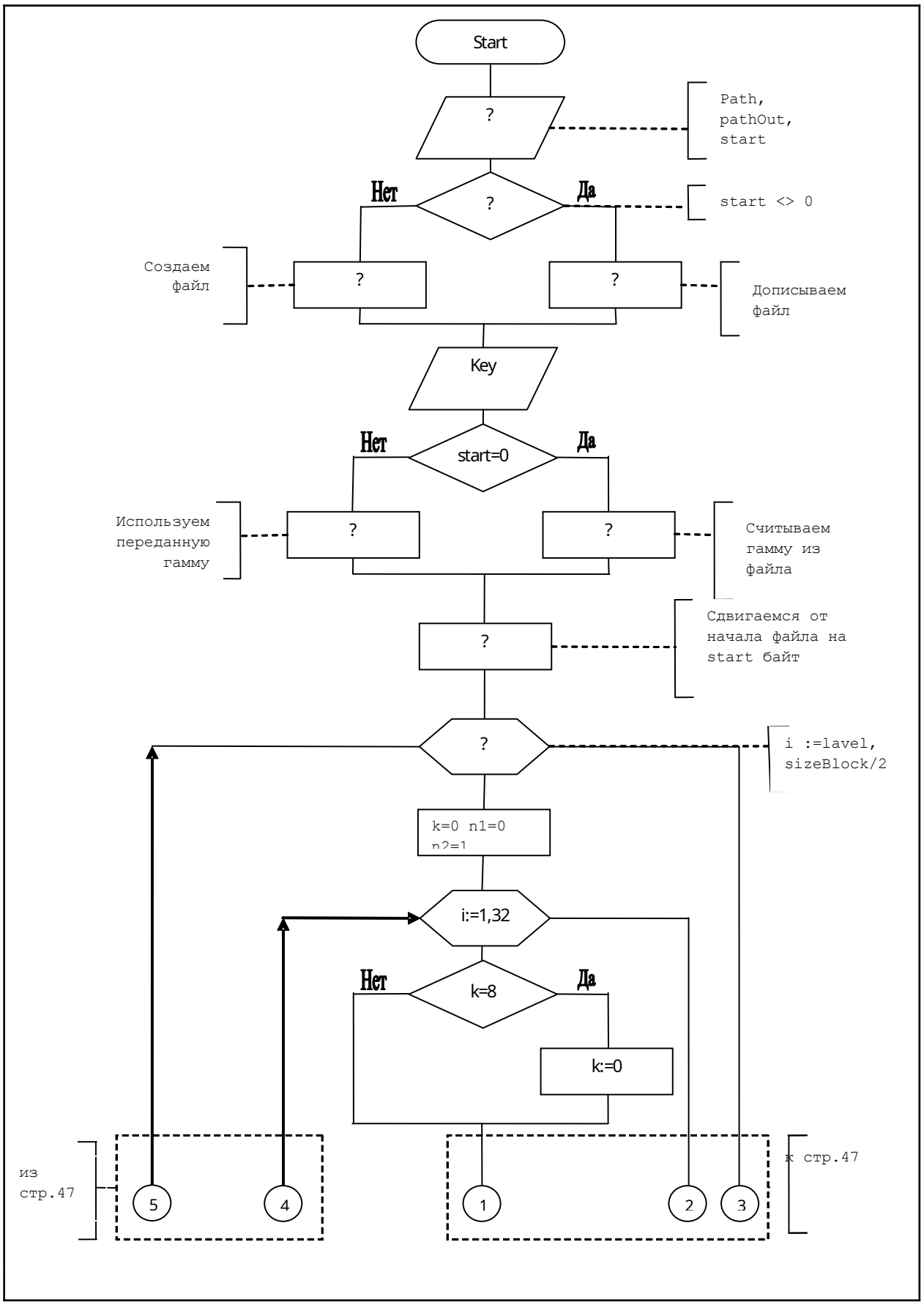


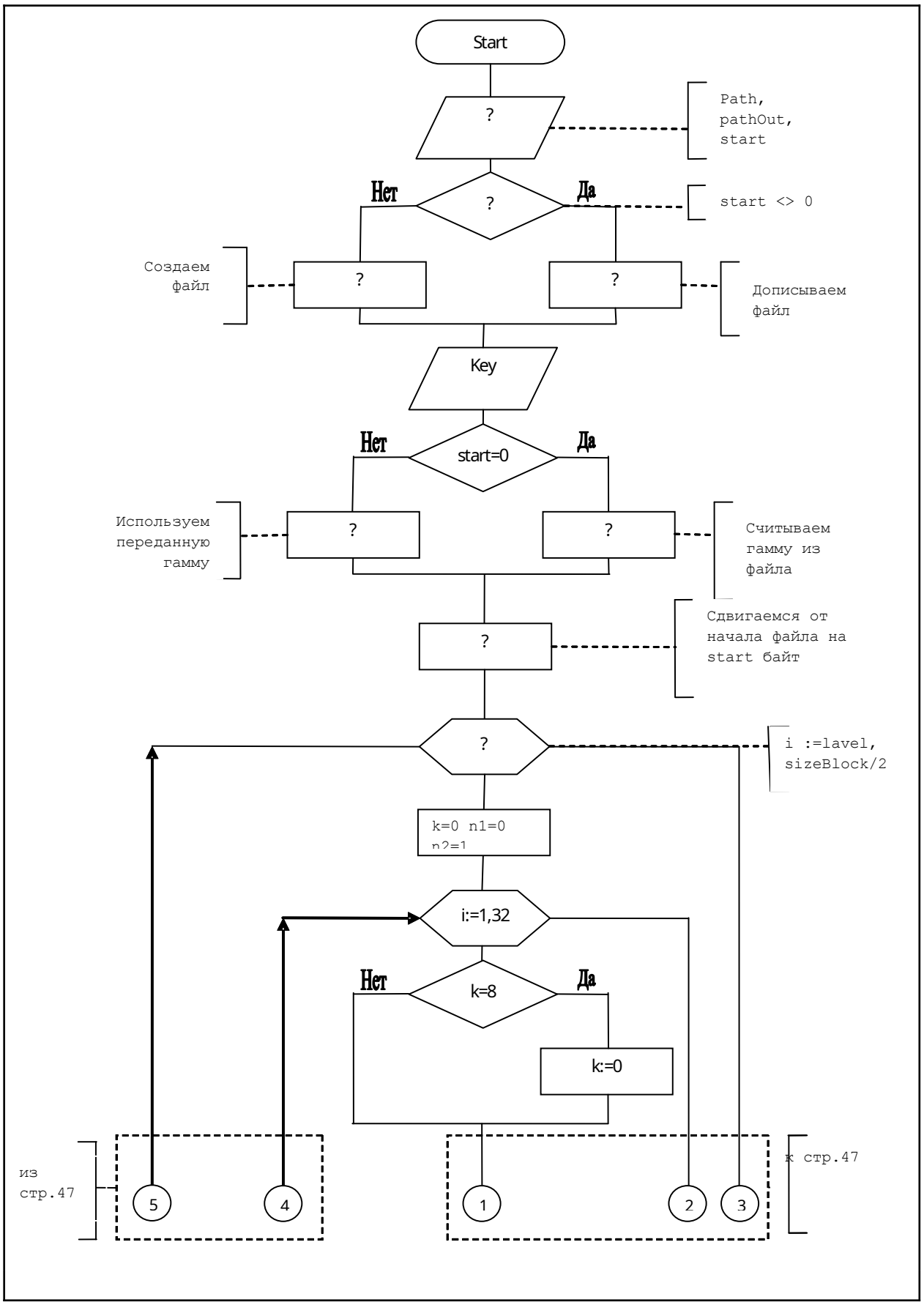


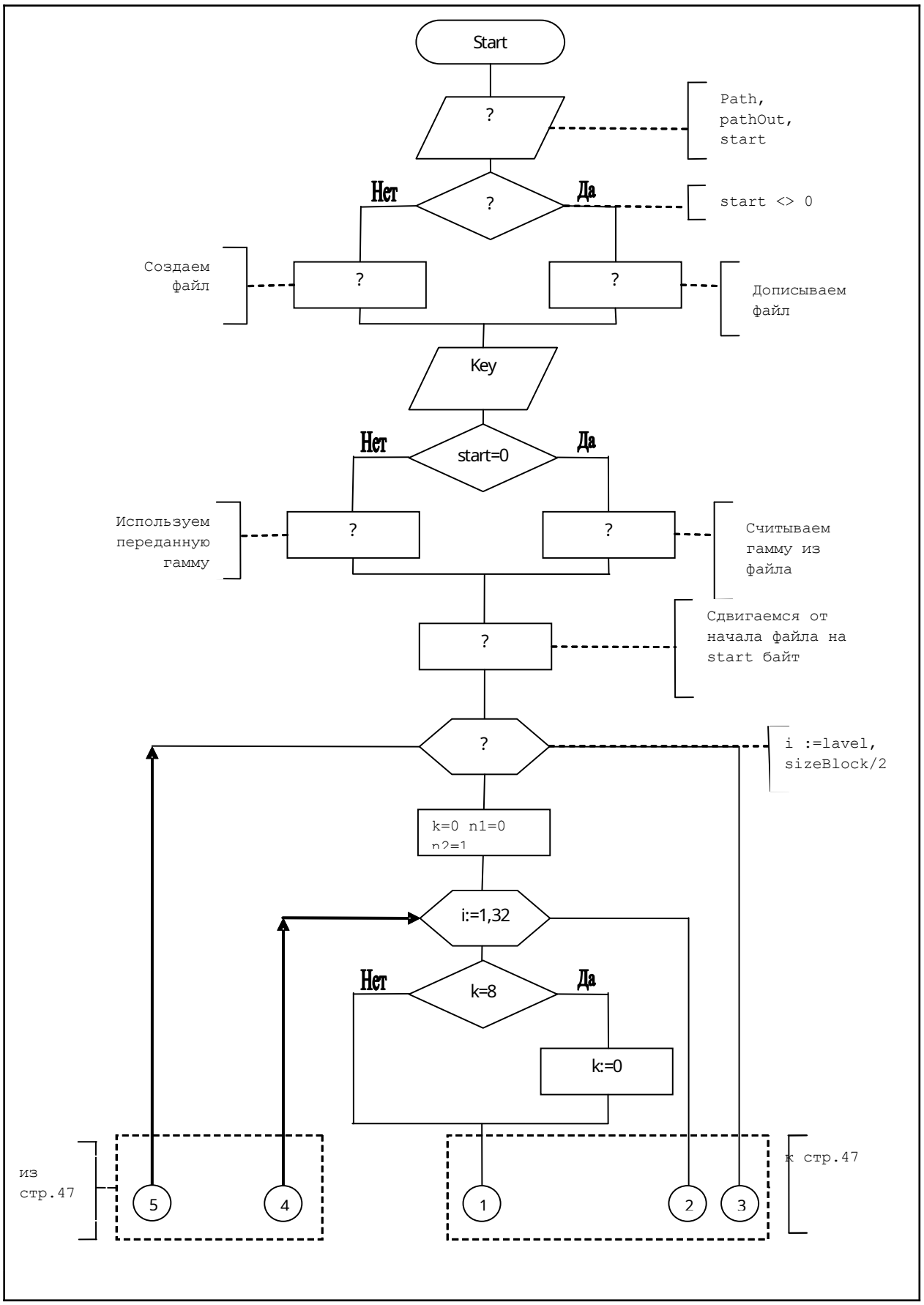
UML-диаграмма

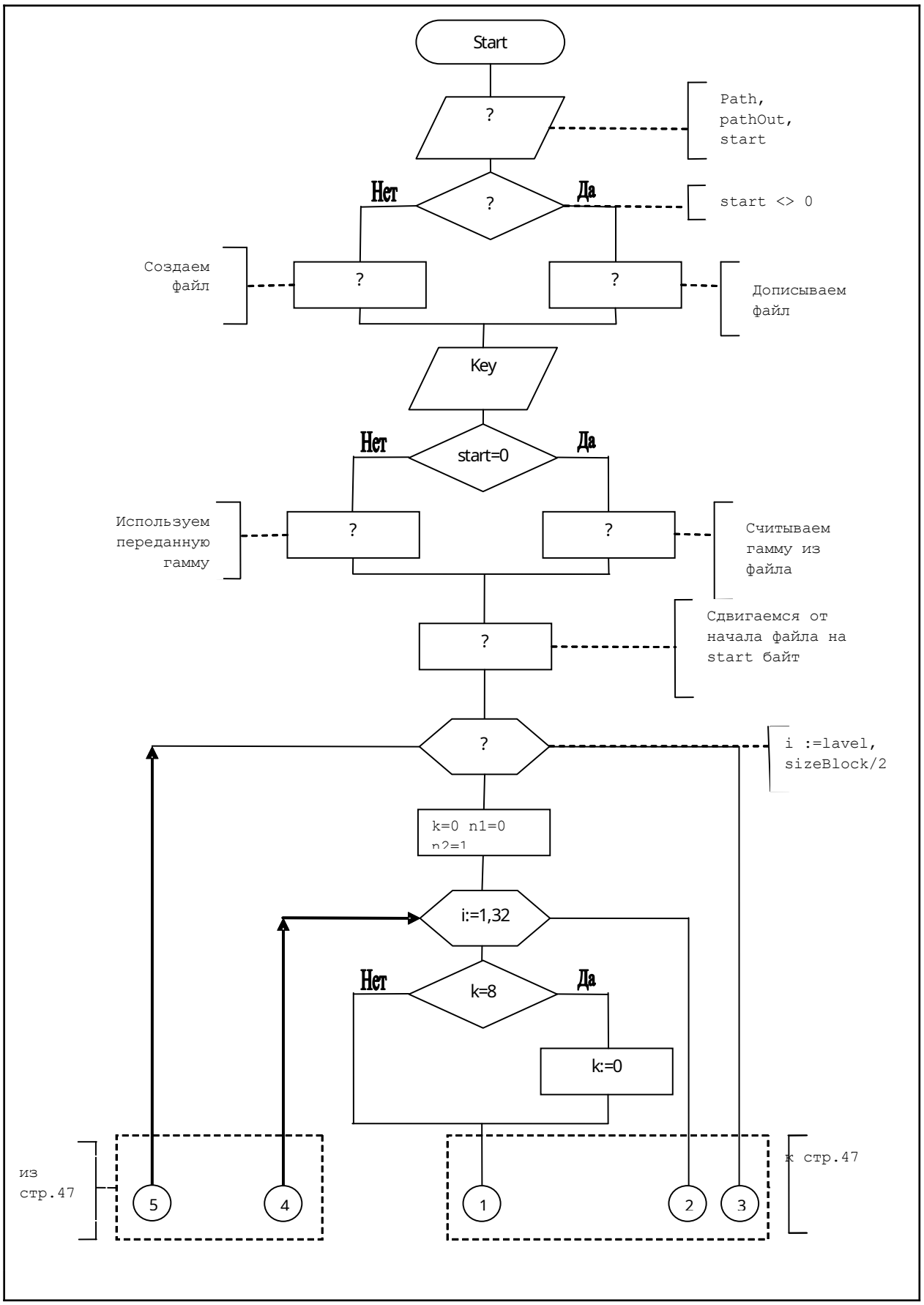


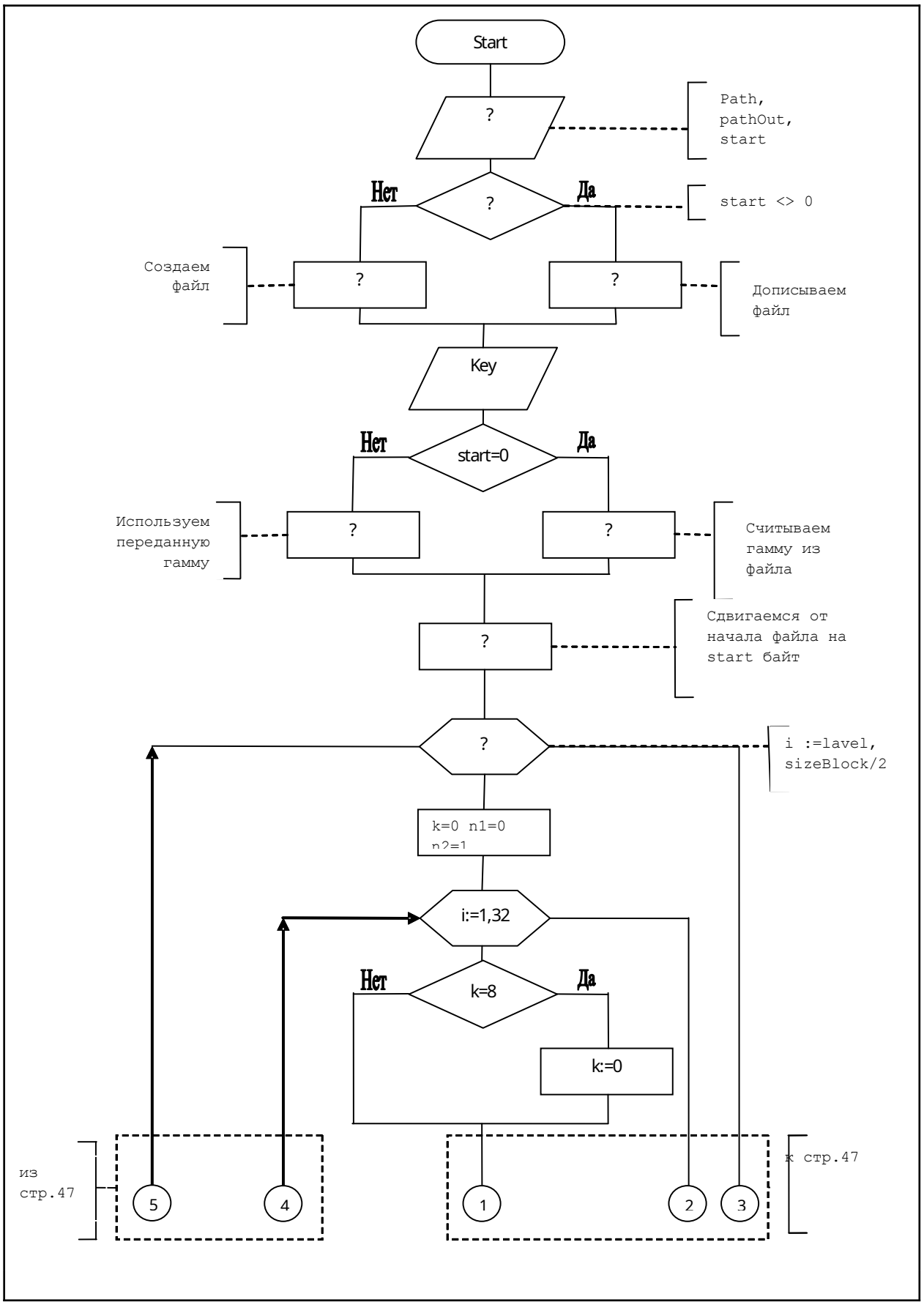
. Диаграмма деятельности алгоритма ГОСТ28147-89

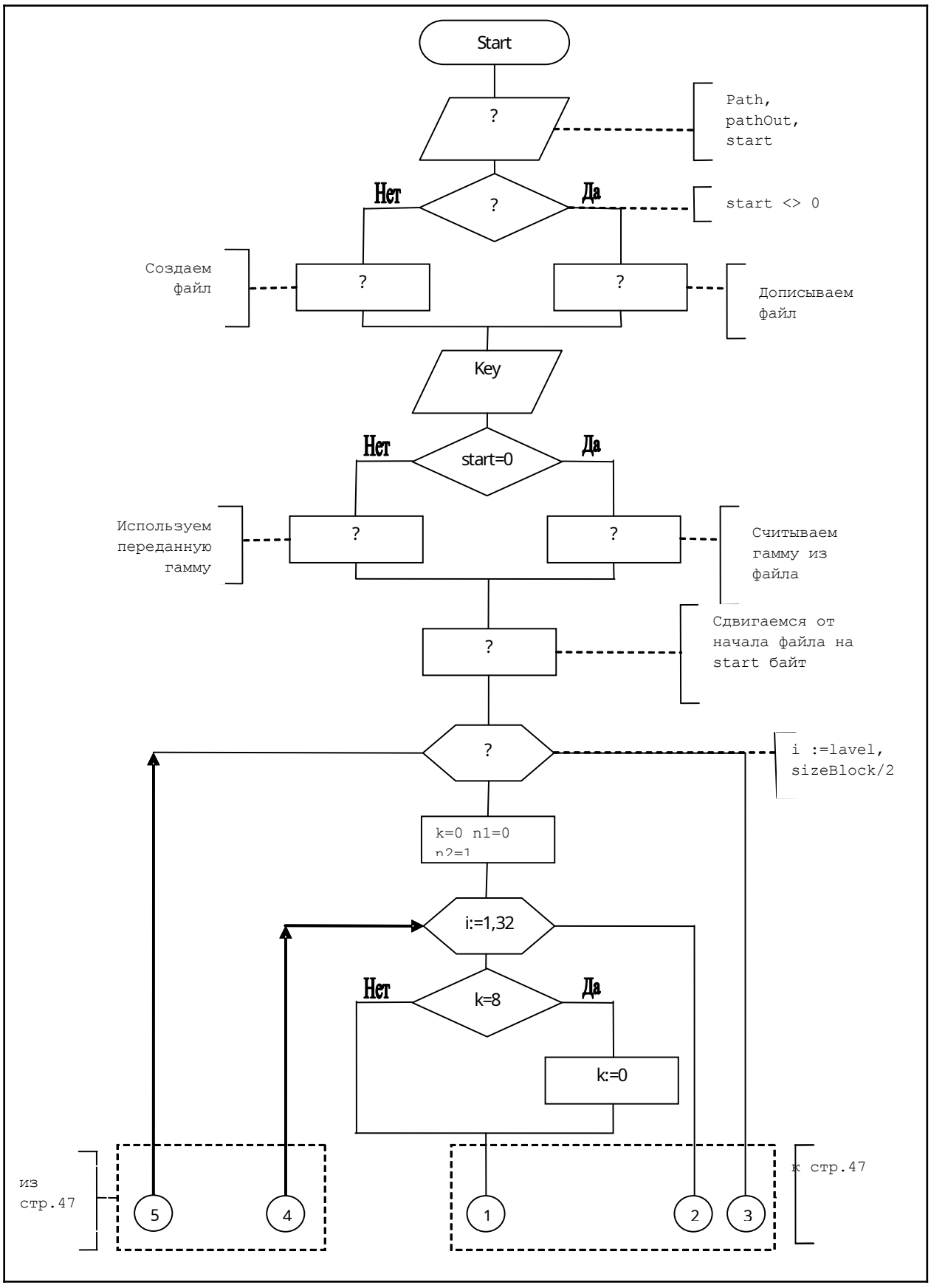

Start
?
Path,
pathOut,
start
Нет
?
Да
start <> 0
Создаем
файл
?
?
Дописываем
файл
Key
Нет
start=0
Да
Используем
переданную
гамму
?
?
Считываем
гамму из
файла
?
Сдвигаемся от
начала файла на
start байт
?
i :=lavel,
sizeBlock/2
k=0 n1=0
n2=1
i:=1,32
Нет
k=8
Да
k:=0
из
стр.47
5
4
1
2
3
к стр.47

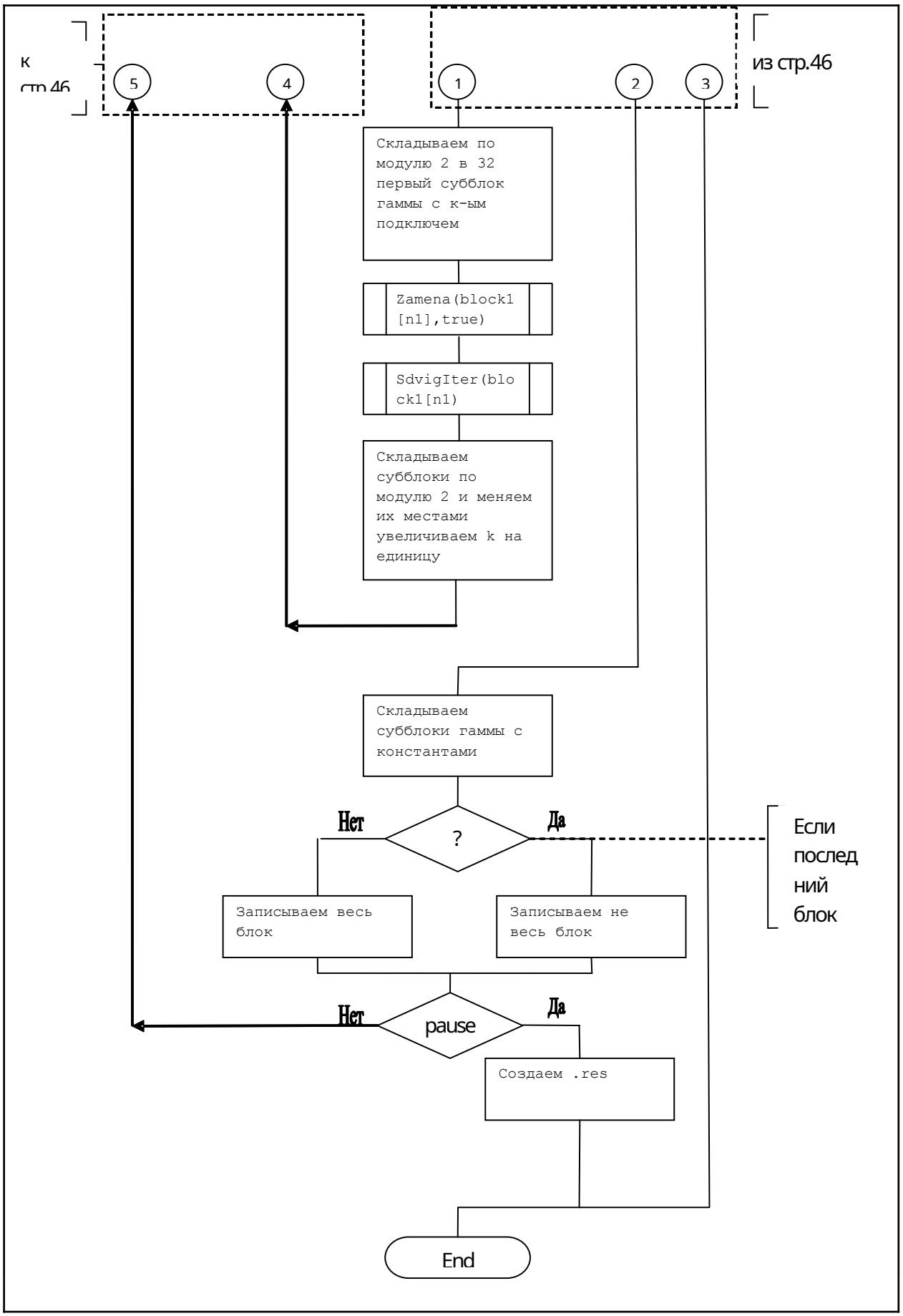

Start
?
Path,
pathOut,
start
Нет
?
Да
start <> 0
Создаем
файл
?
?
Дописываем
файл
Key
Нет
start=0
Да
Используем
переданную
гамму
?
?
Считываем
гамму из
файла
?
Сдвигаемся от
начала файла на
start байт
?
i :=lavel,
sizeBlock/2
k=0 n1=0
n2=1
i:=1,32
Нет
k=8
Да
k:=0
из
стр.47
5
4
1
2
3
к стр.47

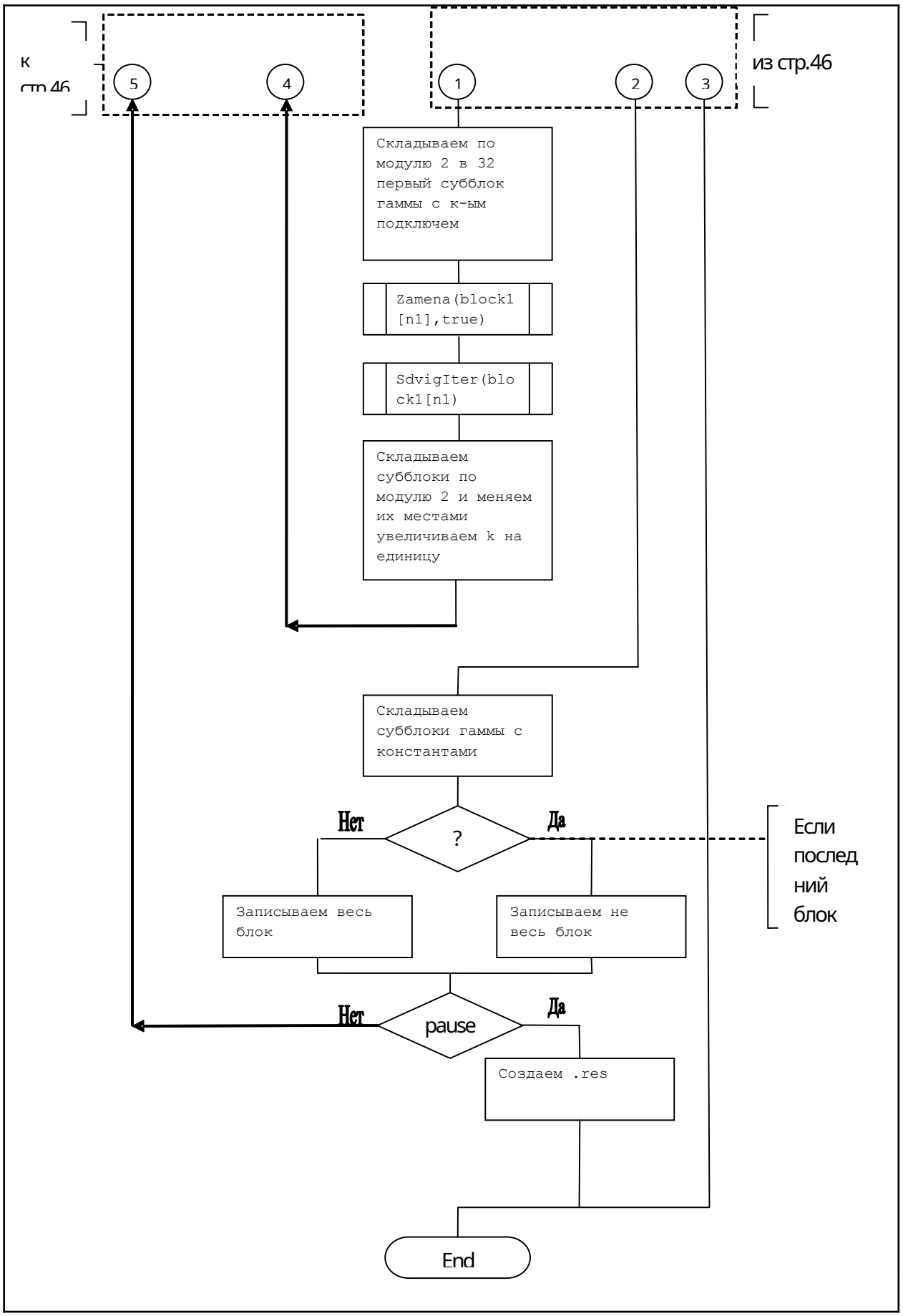

Start
?
Path,
pathOut,
start
Нет
?
Да
start <> 0
Создаем
файл
?
?
Дописываем
файл
Key
Нет
start=0
Да
Используем
переданную
гамму
?
?
Считываем
гамму из
файла
?
Сдвигаемся от
начала файла на
start байт
?
i :=lavel,
sizeBlock/2
k=0 n1=0
n2=1
i:=1,32
Нет
k=8
Да
k:=0
из
стр.47
5
4
1
2
3
к стр.47

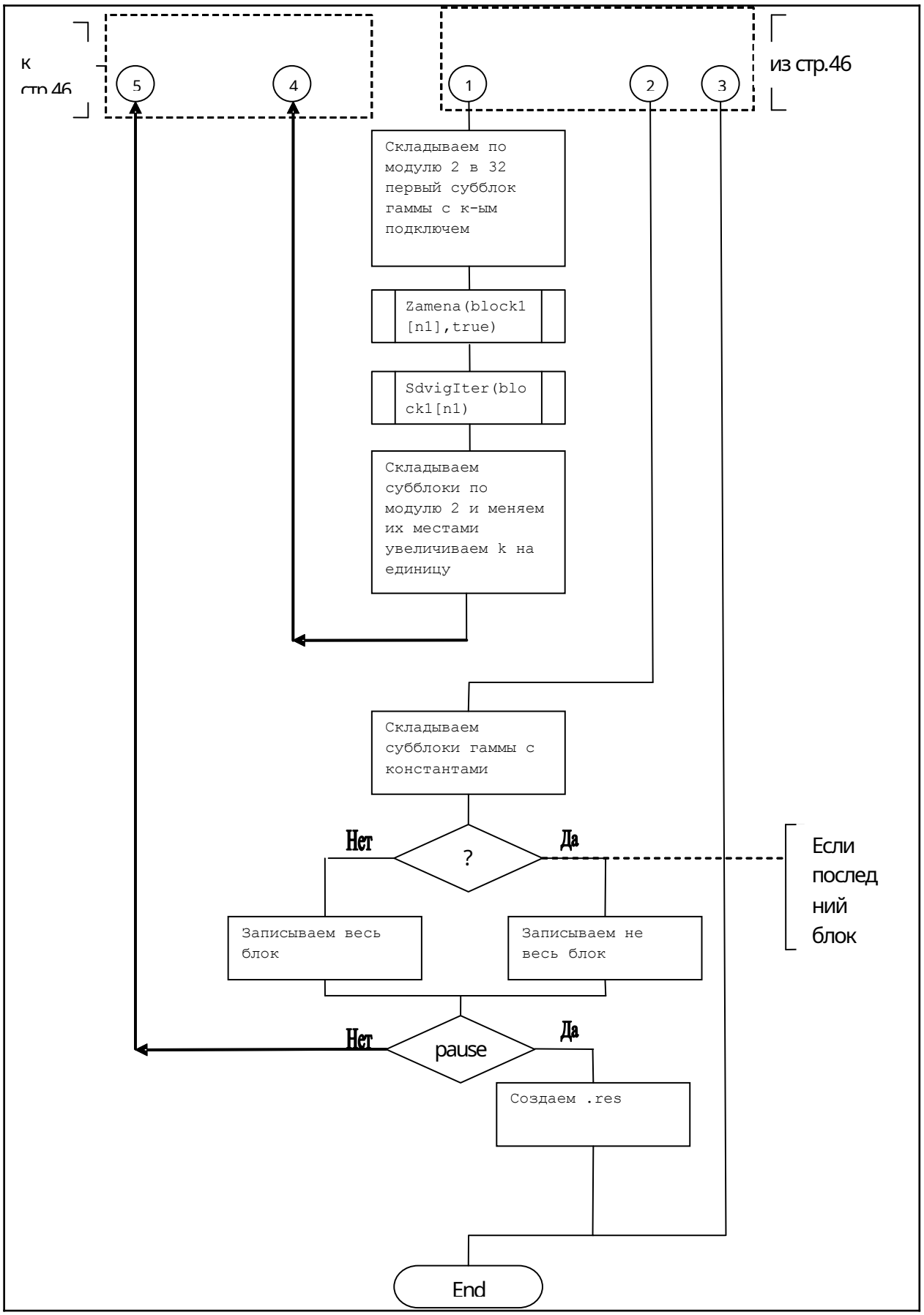

Start
?
Path,
pathOut,
start
Нет
?
Да
start <> 0
Создаем
файл
?
?
Дописываем
файл
Key
Нет
start=0
Да
Используем
переданную
гамму
?
?
Считываем
гамму из
файла
?
Сдвигаемся от
начала файла на
start байт
?
i :=lavel,
sizeBlock/2
k=0 n1=0
n2=1
i:=1,32
Нет
k=8
Да
k:=0
из
стр.47
5
4
1
2
3
к стр.47

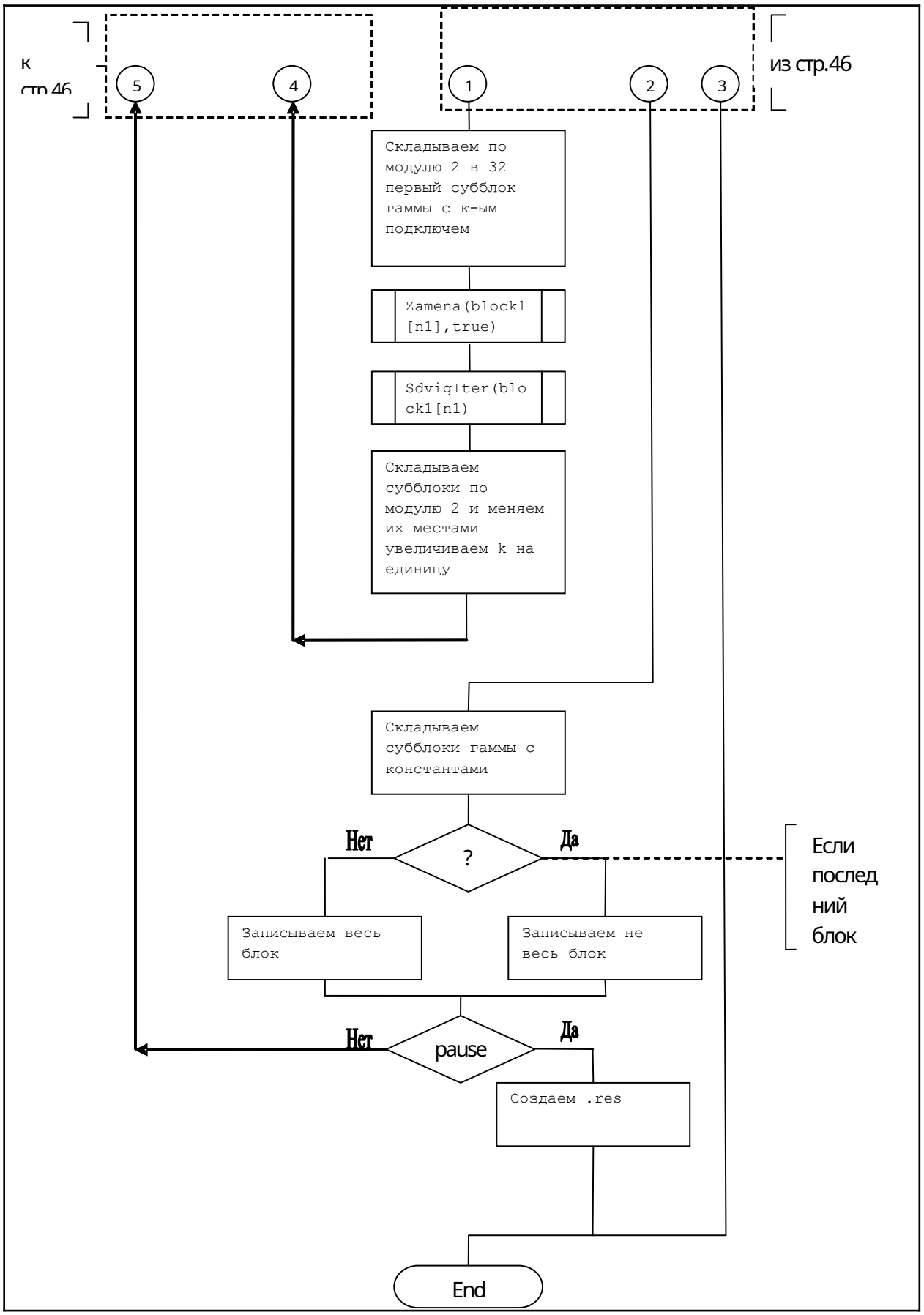

Start
?
Path,
pathOut,
start
Нет
?
Да
start <> 0
Создаем
файл
?
?
Дописываем
файл
Key
Нет
start=0
Да
Используем
переданную
гамму
?
?
Считываем
гамму из
файла
?
Сдвигаемся от
начала файла на
start байт
?
i :=lavel,
sizeBlock/2
k=0 n1=0
n2=1
i:=1,32
Нет
k=8
Да
k:=0
из
стр.47
5
4
1
2
3
к стр.47

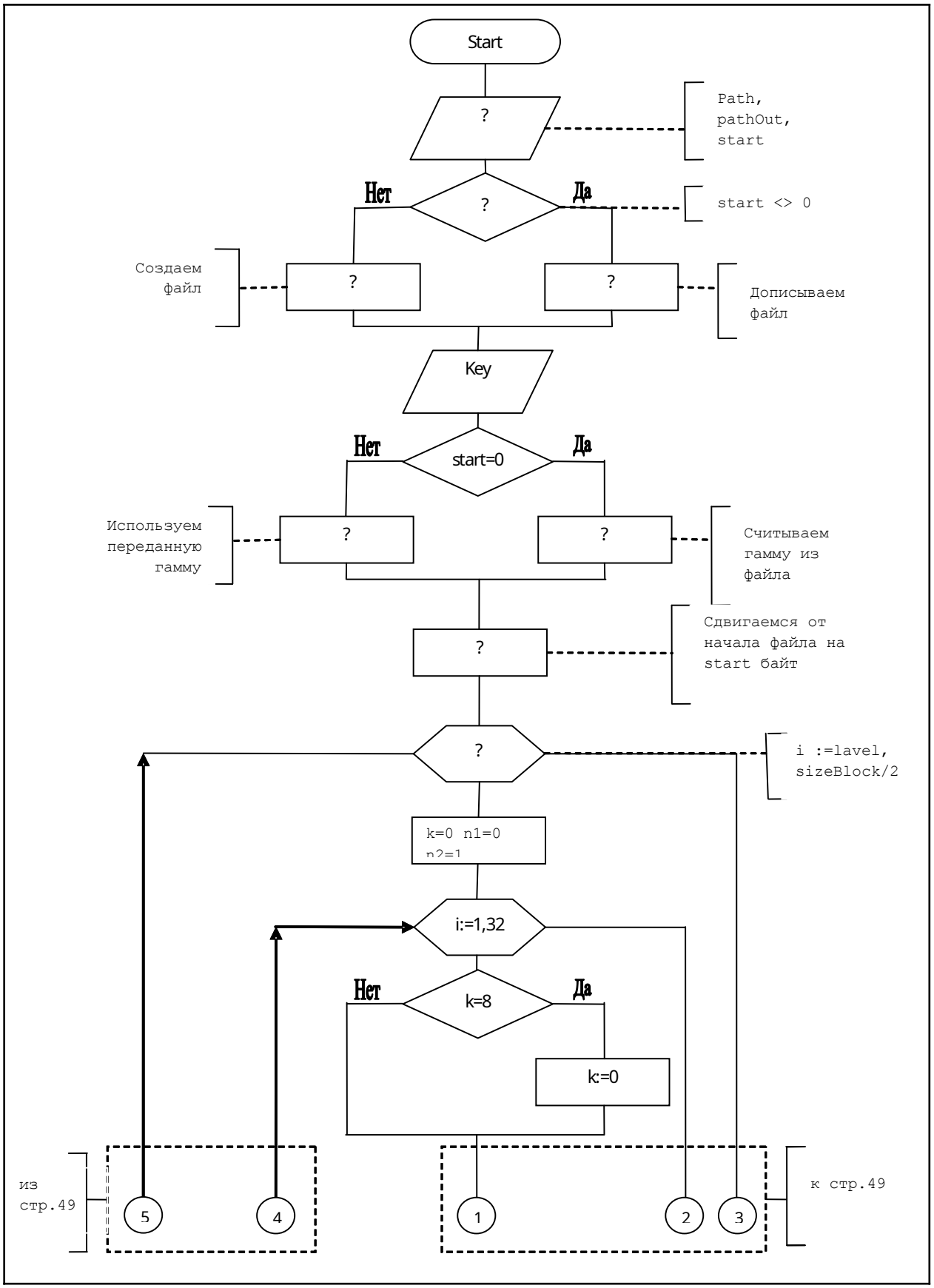

Start
?
Path,
pathOut,
start
Нет
?
Да
start <> 0
Создаем
файл
?
?
Дописываем
файл
Key
Нет
start=0
Да
Используем
переданную
гамму
?
?
Считываем
гамму из
файла
?
Сдвигаемся от
начала файла на
start байт
?
i :=lavel,
sizeBlock/2
k=0 n1=0
n2=1
i:=1,32
Нет
k=8
Да
k:=0
из
стр.47
5
4
1
2
3
к стр.47

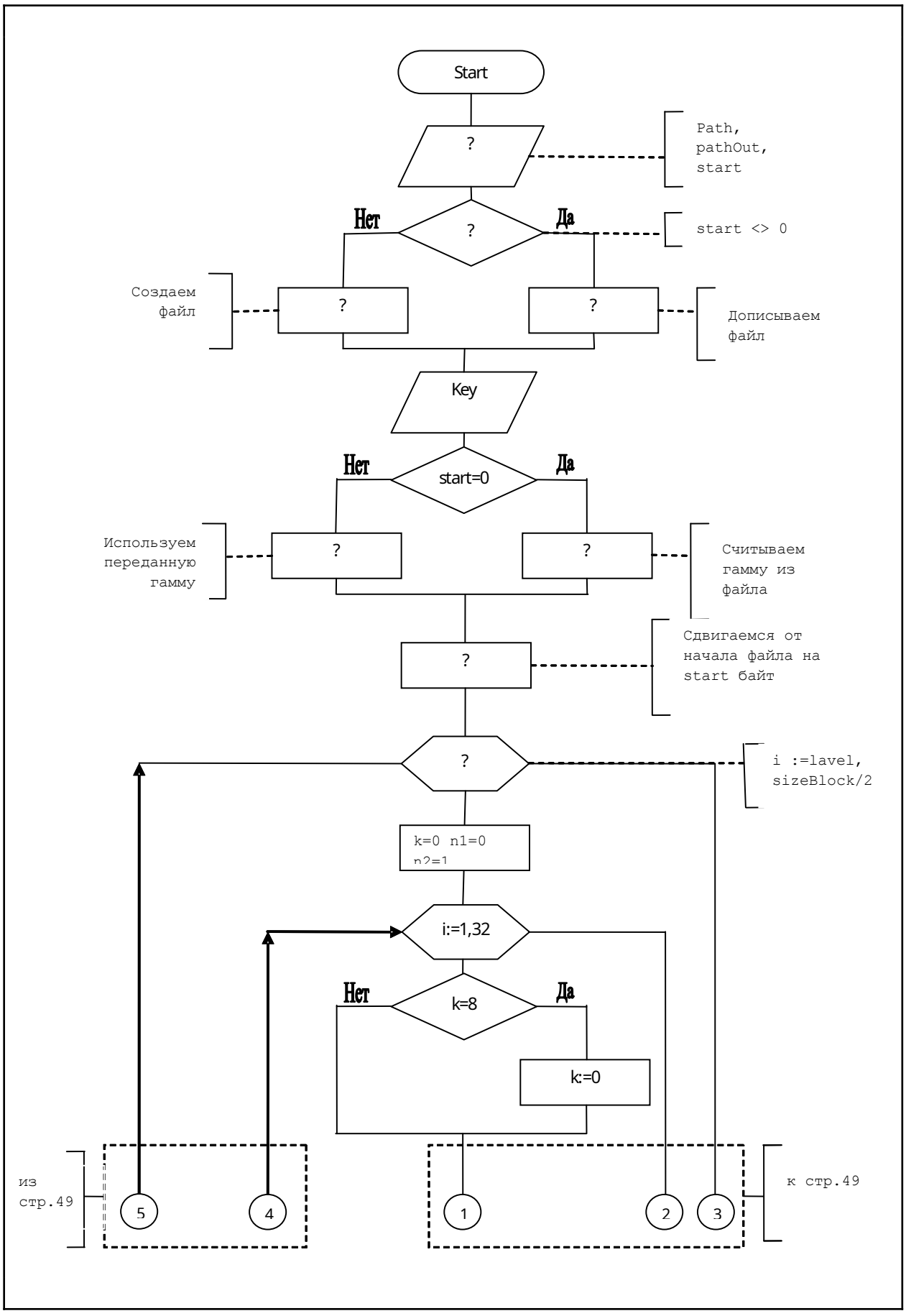

к стр.46
5
4
1
2
3
из стр.46
Складываем по модулю 2 в 32 первый субблок гаммы с к-ым подключем
Zamena(block1[n1],true)
SdvigIter(block1[n1)
Складываем субблоки по модулю 2 и меняем их местами увеличиваем k на единицу
Складываем субблоки гаммы с константами
Нет
?
Да
Если последний блок
Записываем весь блок
Записываем не весь блок
Нет
pause
Да
Создаем .res
End

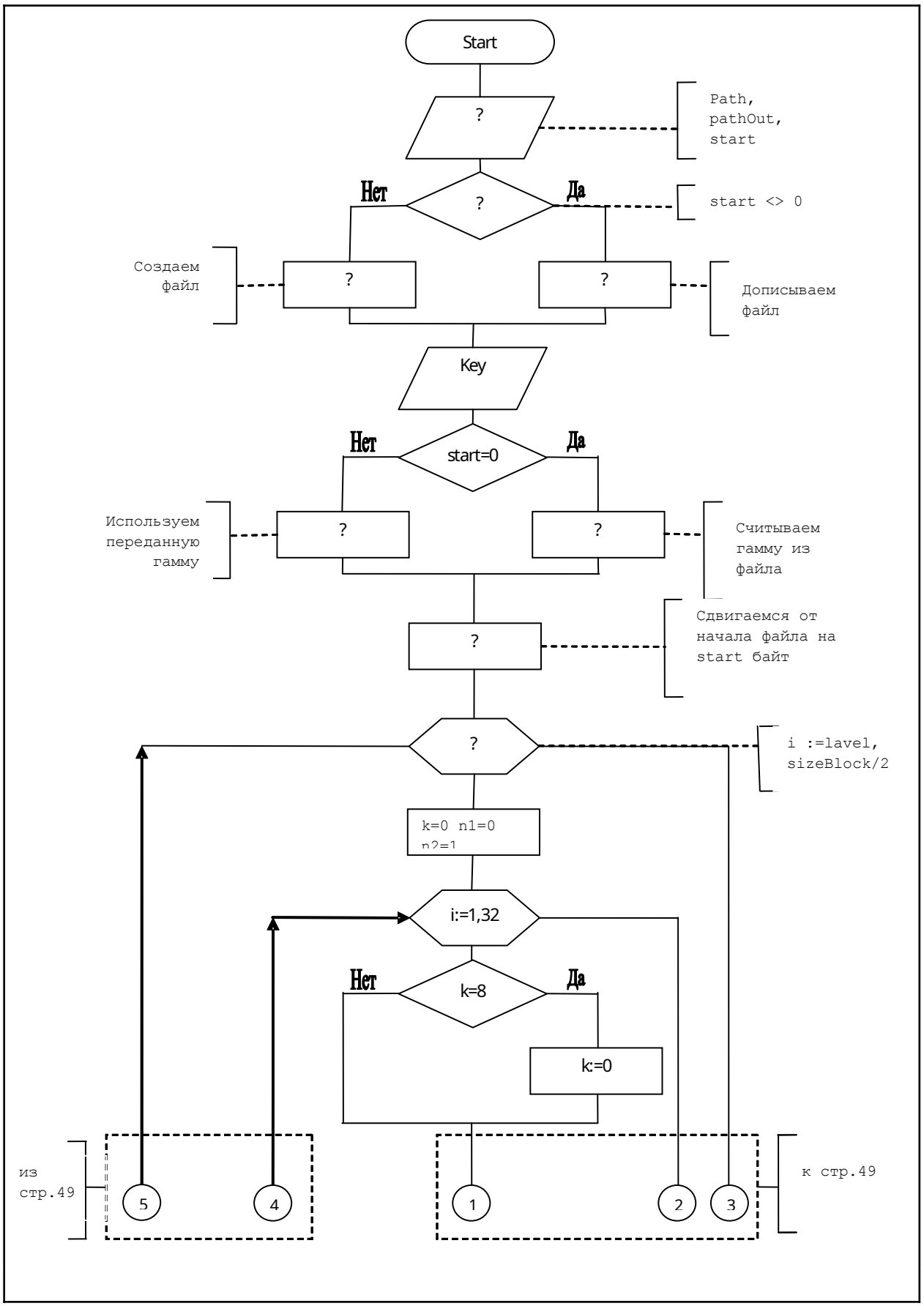

к стр.46
5
4
1
2
3
из стр.46
Складываем по модулю 2 в 32 первый субблок гаммы с к-ым подключем
Zamena(block1[n1],true)
SdvigIter(block1[n1)
Складываем субблоки по модулю 2 и меняем их местами увеличиваем k на единицу
Складываем субблоки гаммы с константами
Нет
?
Да
Если последний блок
Записываем весь блок
Записываем не весь блок
Нет
pause
Да
Создаем .res
End

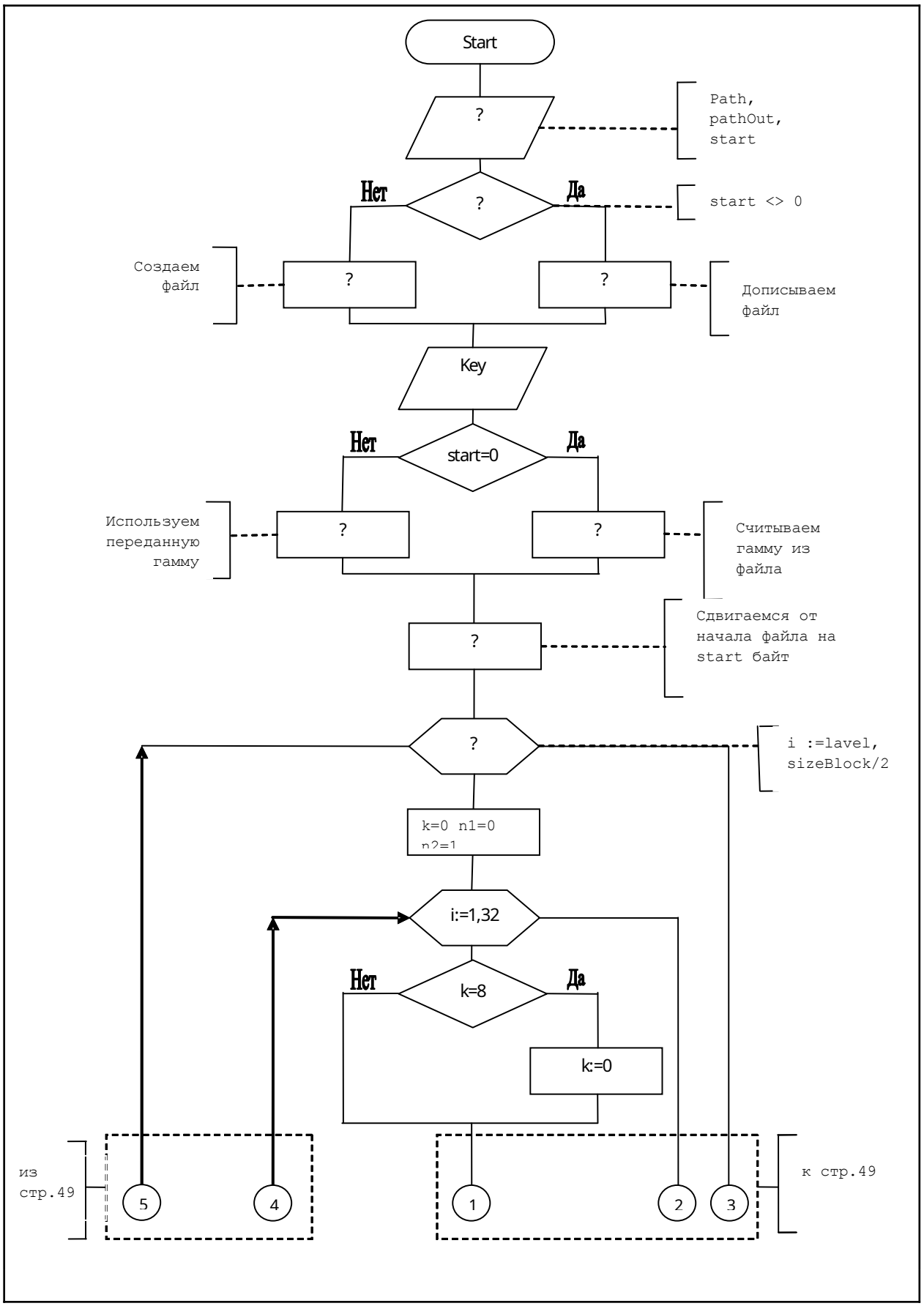

к стр.46
5
4
1
2
3
из стр.46
Складываем по модулю 2 в 32 первый субблок гаммы с к-ым подключем
Zamena(block1[n1],true)
SdvigIter(block1[n1)
Складываем субблоки по модулю 2 и меняем их местами увеличиваем k на единицу
Складываем субблоки гаммы с константами
Нет
?
Да
Если последний блок
Записываем весь блок
Записываем не весь блок
Нет
pause
Да
Создаем .res
End

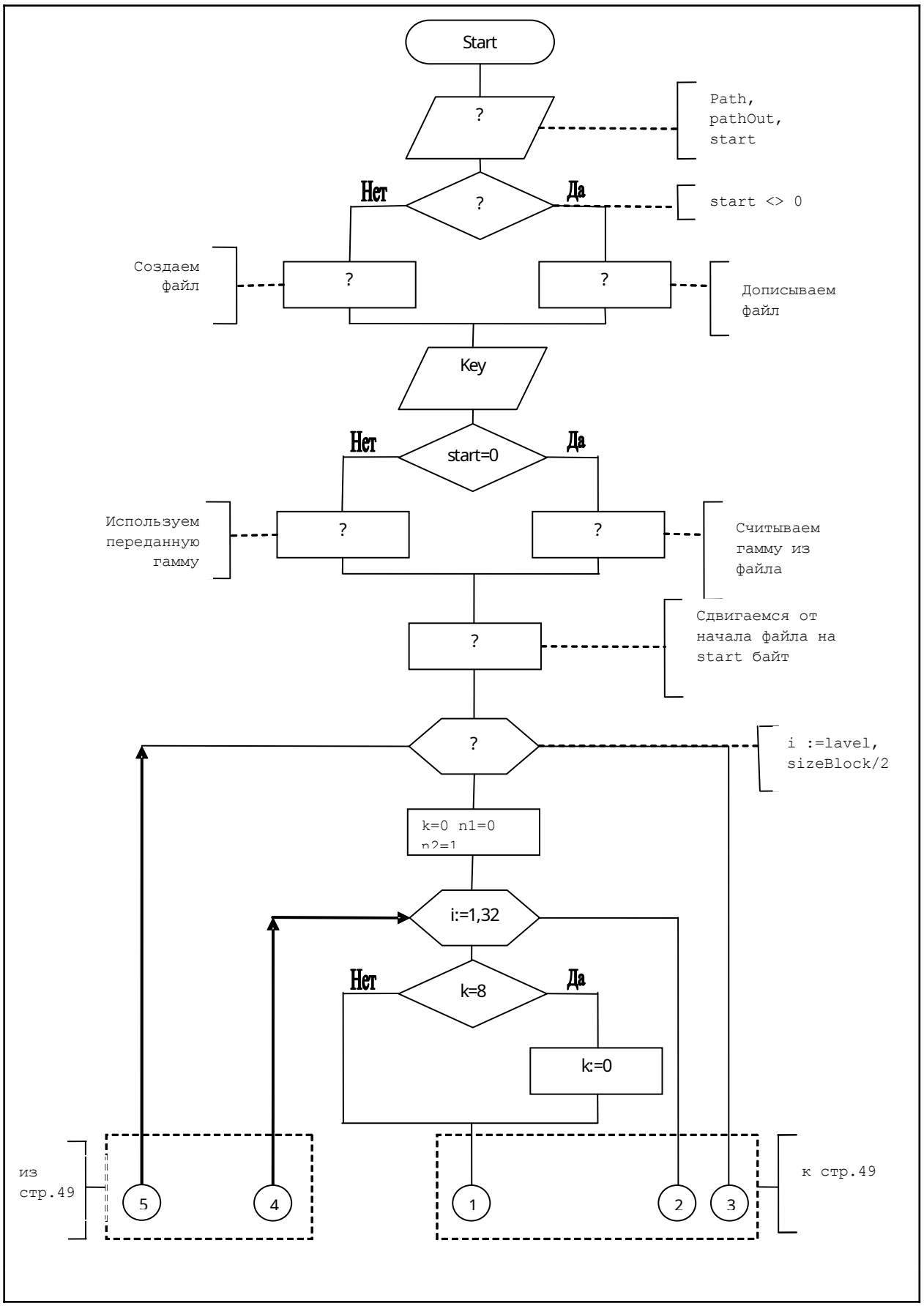

к стр.46
5
4
1
2
3
из стр.46
Складываем по модулю 2 в 32 первый субблок гаммы с к-ым подключем
Zamena(block1[n1],true)
SdvigIter(block1[n1)
Складываем субблоки по модулю 2 и меняем их местами увеличиваем k на единицу
Складываем субблоки гаммы с константами
Нет
?
Да
Если последний блок
Записываем весь блок
Записываем не весь блок
Нет
pause
Да
Создаем .res
End

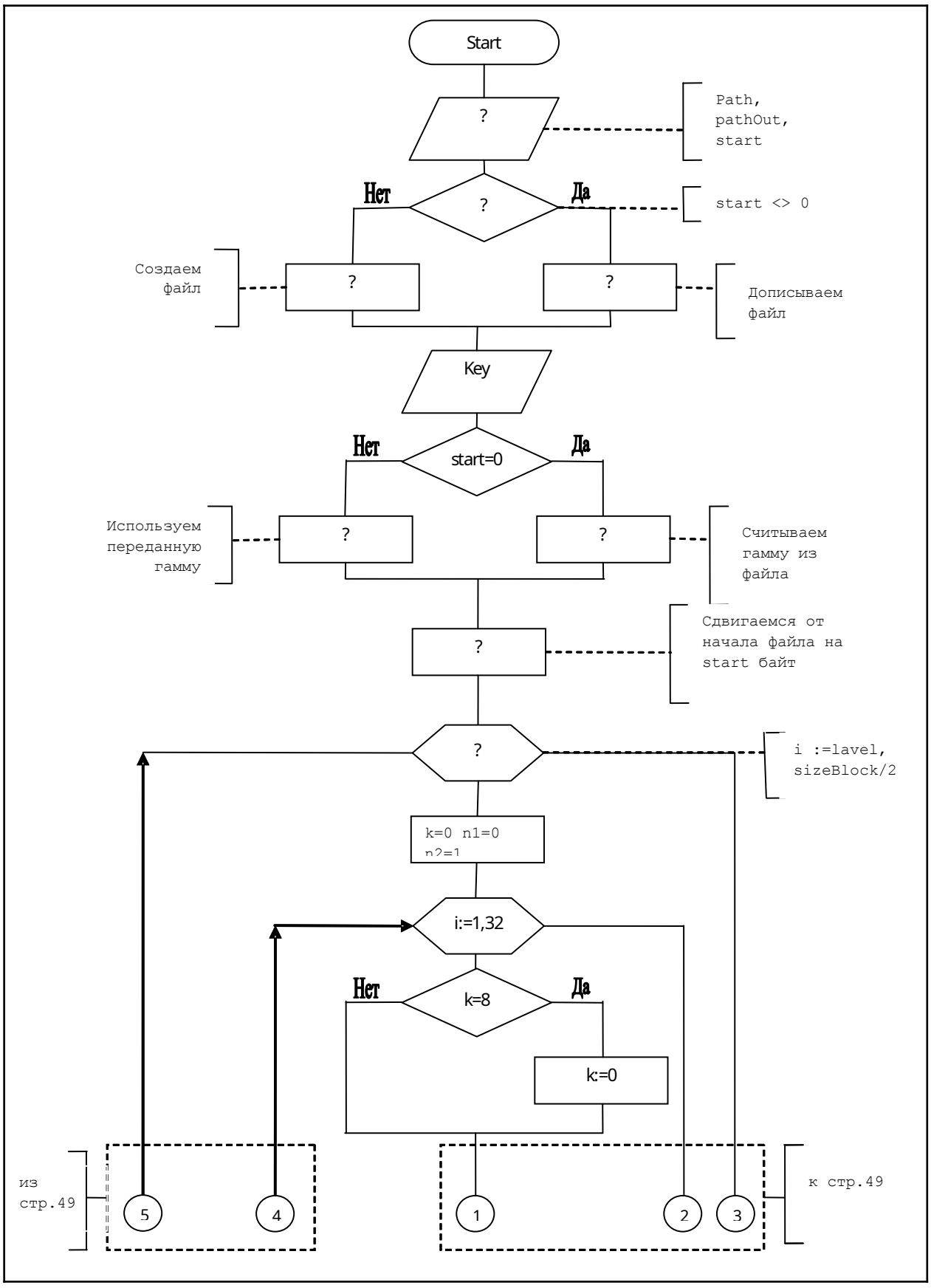

Start
?
Path,
pathOut,
start
Нет
?
Да
start <> 0
Создаем
файл
?
?
Дописываем
файл
Key
Нет
start=0
Да
Используем
переданную
гамму
?
?
Считываем
гамму из
файла
?
Сдвигаемся от
начала файла на
start байт
?
i :=lavel,
sizeBlock/2
k=0 n1=0
n2=1
i:=1,32
Нет
k=8
Да
k:=0
из
стр.49
5
4
1
2
3
к стр.49

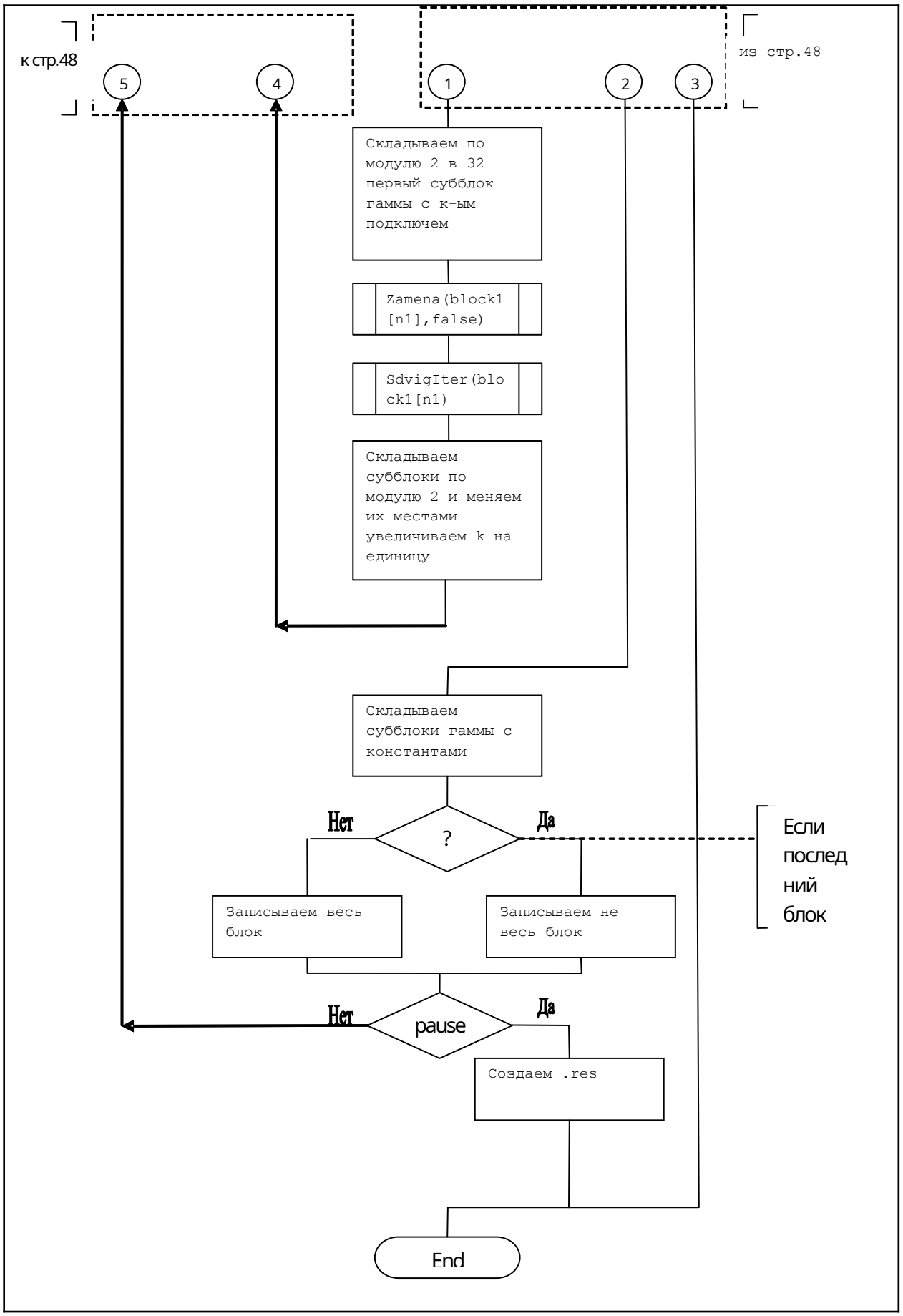

Start
?
Path,
pathOut,
start
Нет
?
Да
start <> 0
Создаем
файл
?
?
Дописываем
файл
Key
Нет
start=0
Да
Используем
переданную
гамму
?
?
Считываем
гамму из
файла
?
Сдвигаемся от
начала файла на
start байт
?
i :=lavel,
sizeBlock/2
k=0 n1=0
n2=1
i:=1,32
Нет
k=8
Да
k:=0
из
стр.49
5
4
1
2
3
к стр.49

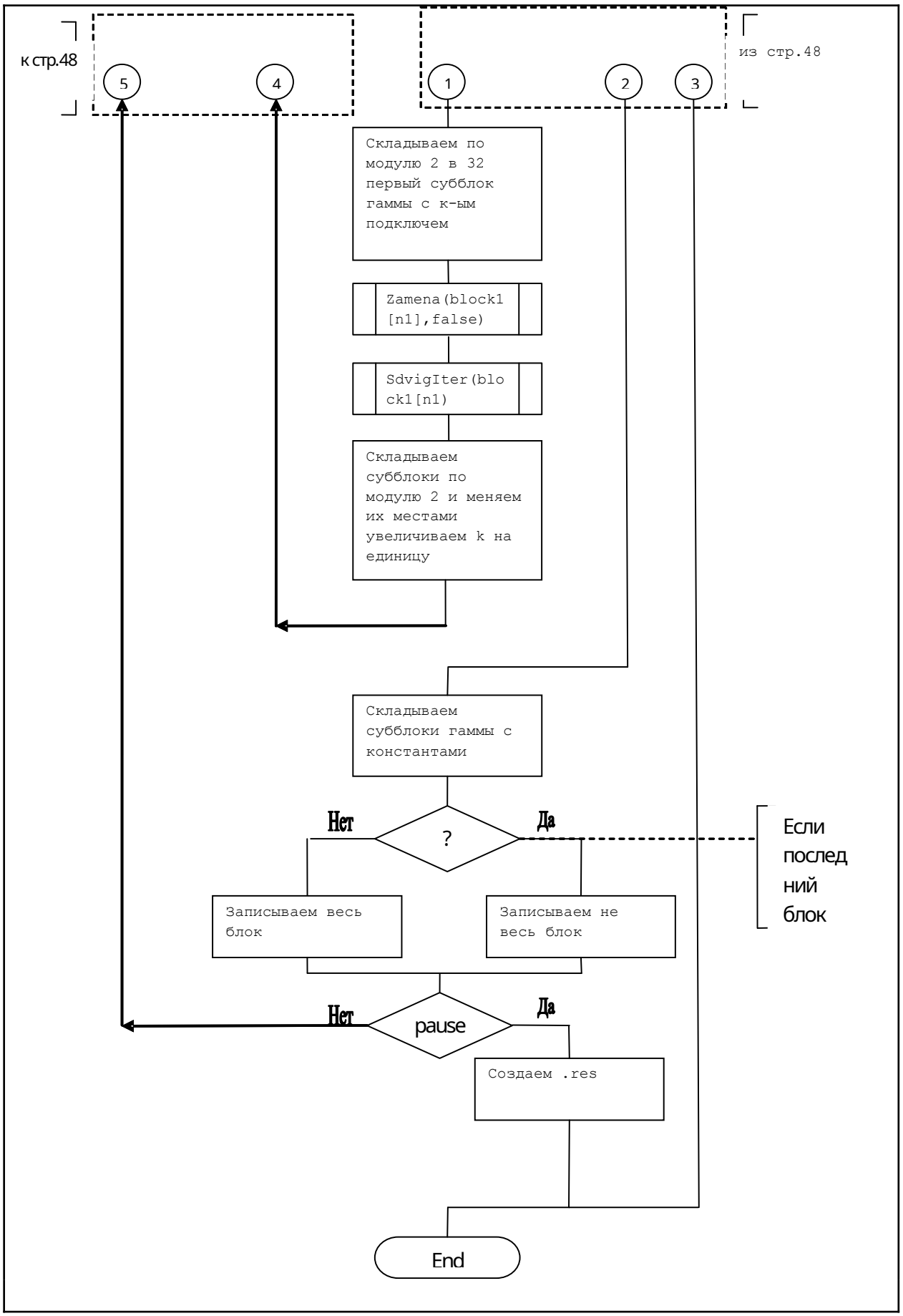

Start
?
Path,
pathOut,
start
Нет
?
Да
start <> 0
Создаем
файл
?
?
Дописываем
файл
Key
Нет
start=0
Да
Используем
переданную
гамму
?
?
Считываем
гамму из
файла
?
Сдвигаемся от
начала файла на
start байт
?
i :=lavel,
sizeBlock/2
k=0 n1=0
n2=1
i:=1,32
Нет
k=8
Да
k:=0
из
стр.49
5
4
1
2
3
к стр.49

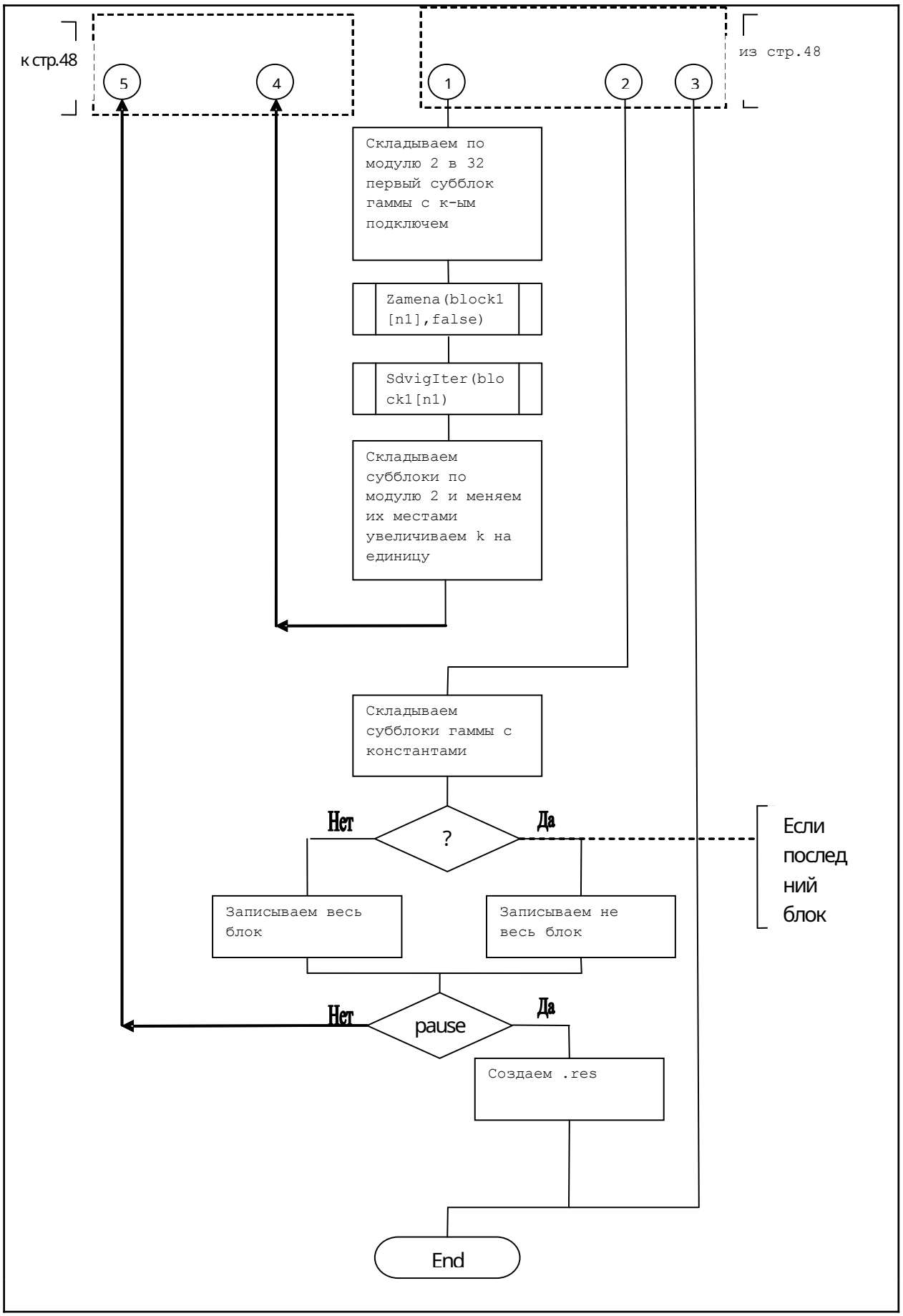

Start
?
Path,
pathOut,
start
Нет
?
Да
start <> 0
Создаем
файл
?
?
Дописываем
файл
Key
Нет
start=0
Да
Используем
переданную
гамму
?
?
Считываем
гамму из
файла
?
Сдвигаемся от
начала файла на
start байт
?
i :=lavel,
sizeBlock/2
k=0 n1=0
n2=1
i:=1,32
Нет
k=8
Да
k:=0
из
стр.49
5
4
1
2
3
к стр.49

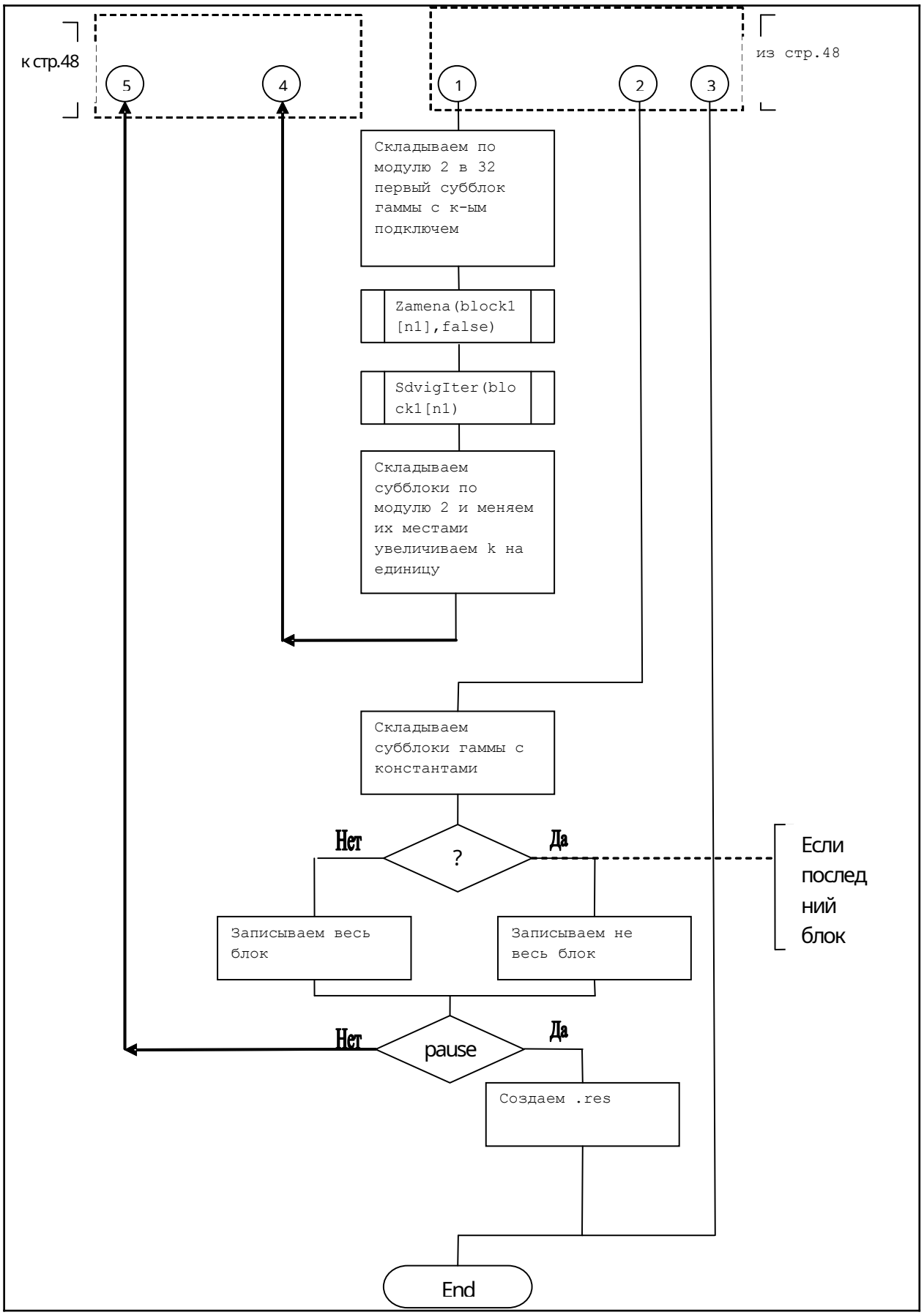

Start
?
Path,
pathOut,
start
Нет
?
Да
start <> 0
Создаем
файл
?
?
Дописываем
файл
Key
Нет
start=0
Да
Используем
переданную
гамму
?
?
Считываем
гамму из
файла
?
Сдвигаемся от
начала файла на
start байт
?
i :=lavel,
sizeBlock/2
k=0 n1=0
n2=1
i:=1,32
Нет
k=8
Да
k:=0
из
стр.49
5
4
1
2
3
к стр.49


Start
?
Path,
pathOut,
start
Нет
?
Да
start <> 0
Создаем
файл
?
?
Дописываем
файл
Key
Нет
start=0
Да
Используем
переданную
гамму
?
?
Считываем
гамму из
файла
?
Сдвигаемся от
начала файла на
start байт
?
i :=lavel,
sizeBlock/2
k=0 n1=0
n2=1
i:=1,32
Нет
k=8
Да
k:=0
из
стр.49
5
4
1
2
3
к стр.49


к стр.48
5
4
1
2
3
из стр.48
Складываем по модулю 2 в 32 первый субблок гаммы с к-ым подключем
Zamena(block1[n1],false)
SdvigIter(block1[n1)
Складываем субблоки по модулю 2 и меняем их местами увеличиваем k на единицу
Складываем субблоки гаммы с константами
Нет
?
Да
Если последний блок
Записываем весь блок
Записываем не весь блок
Нет
pause
Да
Создаем .res
End


к стр.48
5
4
1
2
3
из стр.48
Складываем по модулю 2 в 32 первый субблок гаммы с к-ым подключем
Zamena(block1[n1],false)
SdvigIter(block1[n1)
Складываем субблоки по модулю 2 и меняем их местами увеличиваем k на единицу
Складываем субблоки гаммы с константами
Нет
?
Да
Если последний блок
Записываем весь блок
Записываем не весь блок
Нет
pause
Да
Создаем .res
End


к стр.48
из стр.48
5
4
1
2
3
Складываем по модулю 2 в 32 первый субблок гаммы с к-ым подключем
Zamena(block1[n1],false)
SdvigIter(block1[n1)
Складываем субблоки по модулю 2 и меняем их местами увеличиваем k на единицу
Складываем субблоки гаммы с константами
Нет
?
Да
Если последний блок
Записываем весь блок
Записываем не весь блок
Нет
pause
Да
Создаем .res
End


к стр.48
из стр.48
5
4
1
2
3
Складываем по модулю 2 в 32 первый субблок гаммы с к-ым подключем
Zamena(block1[n1],false)
SdvigIter(block1[n1)
Складываем субблоки по модулю 2 и меняем их местами увеличиваем k на единицу
Складываем субблоки гаммы с константами
Нет
?
Да
Если последний блок
Записываем весь блок
Записываем не весь блок
Нет
pause
Да
Создаем .res
End