Московский государственный технический университет им.Н.Э.Баумана
Калужский филиал
Приборостроительный факультет
Кафедра П2-КФ

Отчет

по лабораторной работе № 2

«Реализация и исследование решения дифференциального уравнения методом Эйлера на языке ассемблера с использовани-ем команд CPU и FPU»

по курсу

«Теория и проектирование СВК»

Студент : Комиссаров А.В., гр.ЭВМ-102

Преподаватель : доцент Максимов А.В.

Калуга, 1999 г.

Задание (вариант № 25)

- Разработать математический и машинный алгоритм решения дифференциального уравнения методом Эйлера (два варианта: с использованием основных команд процессора и блока FPU);

- Осуществить сопоставление вариантов по точности, по времени исполнения и объему занимаемой памяти. Определить максимальную глубину стека для варианта, использующего FPU. Для сравнения вариантов взять 3 различных шага. Построить гистограммы дисперсий ошибок для различных шагов интегрирования.

- **Исходные данные** :

  - уравнение : $y''' - 2y'' + y = 0;$
  - интервал интегрирования : 0…1,2;
  - начальные условия :
    $y(0) = 0{,}5; y'(0) = 0; y''(0) = 0; y'''(0) = -y(0) + 2y''(0) = -0{,}5;$
  - шаг интегрирования h принять равным
    $2^{-4}(0{,}0625); 2^{-5}(0{,}03125); 2^{-6}(0{,}015625).$

Аналитическое (точное) решение заданного дифференциального уравнения

Решение данного однородного дифференциального уравнения имеет следующий вид:

$y(x) = C_1 e^{\lambda_1 x} + C_2 e^{\lambda_2 x} + C_3 e^{\lambda_3 x}$, где λ_1, λ_2 и λ_3 - корни характеристического уравнения $\lambda^3 - 2\lambda^2 + 1 = 0$. Решая уравнение, получим: $\lambda_1 = 1$; $\lambda_2 = 0{,}5 + \frac{\sqrt{5}}{2} \approx 1{,}618$; $\lambda_3 = 0{,}5 - \frac{\sqrt{5}}{2} \approx -0{,}618$.

Коэффициенты C_1, C_2 и C_3 определяются исходя из начальных условий. Имеем следующую систему уравнений :

$$y(0) = C_1 + C_2 + C_3 = 0{,}5;$$
$$y'(0) = C_1 + C_2\left(0{,}5 + \frac{\sqrt{5}}{2}\right) + C_3\left(0{,}5 - \frac{\sqrt{5}}{2}\right) = 0;$$
$$y''(0) = C_1 + C_2\left(0{,}5 + \frac{\sqrt{5}}{2}\right)^2 + C_3\left(0{,}5 - \frac{\sqrt{5}}{2}\right)^2 = 0.$$

Решив эту систему, получим следующие значения постоянных :
$C_1 = 0{,}5$; $C_2 \approx -0.2236$; $C_3 \approx 0.2236$.
В результате искомое решение запишется следующим образом :

$$y(x) \cong 0{,}5e^{x} - 0{,}2236e^{1{,}618x} + 0{,}2236e^{-0{,}618x}.$$

Определим абсолютно максимальные значения для решения и первых трех его производных на заданном интервале (так как все эти функции – монотонно убывающие, то максимальное по модулю значение следует искать на одном из концов интервала) :

$|y|_{max} = |y(0)| = 0{,}5;$ $|y'|_{max} = |y'(1{,}2)| \approx 0{,}9276;$ $|y''|_{max} = |y''(1{,}2)| \approx 2{,}3797;$
$|y'''|_{max} = |y'''(1{,}2)| = 4{,}9673.$

Математический алгоритм

Введем следующие обозначения: $y_1 = y'(x); y_2 = y''(x); y_3 = y'''(x).$
Для решения д.у. весь интервал разбивается на n участков с величиной шага h. Решение для каждого участка можно найти по следующим формулам:

$y_{3,k} = 2y_{2,k} - y_k$;
$y_{2,k+1} = y_{2,k} + h\, y_{3,k}$;
$y_{1,k+1} = y_{1,k} + h\, y_{2,k}$;
$y_{k+1} = y_k + h\, y_{1,k}$.

На каждой итерации необходимо отслеживать момент достижения правой границы интервала. $y_{1,k}$, $y_{2,k}$ и y_k являются исходными значениями для данной итерации k. $y_{1,k+1}$, $y_{2,k+1}$ и y_{k+1} будут являться исходными значениями для следующей итерации k+1.

Таким образом, чтобы получить решение д.у. методом Эйлера, необходимо произвести n итераций для вычисления $y_{1,k}$ и $y_{2,k}$, где k=1..n, а $n = \frac{1{,}2}{h}$. Для первой итерации исходными данными будут являться начальные значения.

Составим список используемых простейших арифметических операций:

- $z_1 = h\, y_{1,k}$
- $z_2 = z_1 + y_k$ (z_2 - значение y_{k+1})
- $z_3 = 2 \cdot y_{2,k}$
- $z_4 = z_1 - y_k$ (z_4 – значение $y_{3,k}$.)
- $z_5 = h\, y_{3,k}$
- $z_6 = z_5 + y_{2,k}$ (z_6 - значение $y_{2,k+1}$)
- $z_7 = h\, y_{2,k}$
- $z_8 = z_7 + y_{1,k}$ (z_8 - значение $y_{1,k+1}$)
- $z_9 = x_k + h$ (z_9 - значение x_{k+1}).

Машинный алгоритм и масштабирование

Машинный алгоритм для реализации на FPU в целом совпадает с приведенным математическим алгоритмом, а машинный алгоритм для реализации на CPU подробно рассматривался в лабораторной работе №3 по курсу "Теория и проектирование ЭВМ и систем" за предыдущий семестр. Масштабные соотношения для входных и выходных переменных имеют следующий вид:

$$M_x = 2^{-1} \cdot 2^{15} = 2^{14};\ M_h = 2^3 \cdot 2^{15} = 2^{18};\ M_y = 2^0 \cdot 2^{15} = 2^{15};\ M_{y1} = 2^0 \cdot 2^{15} = 2^{15};$$
$$M_{y2} = 2^{-2} \cdot 2^{15} = 2^{13};\ M_{y3} = 2^{-3} \cdot 2^{15} = 2^{12}.$$

Расчет математического ожидания и дисперсии ошибки

Ошибка между *i*-м значением реализуемой на CPU (FPU) функции и *i*-м значением эталонной функции может быть вычислена следующим образом :

$$E_i = y_i - f_i,$$

где y_i - текущее значение реализуемой функции, а f_i - текущее значение эталонной функции.

Математическое ожидание ошибки реализации можно подсчитать по формуле :

$$M[E] = \frac{\sum_{i=0}^{l-1} E_i}{l},$$

l– количество точек, в которых вычисляются значения математического ожидания, это значение на единицу больше, чем количество подынтервалов *n*, на которые разбивается весь интервал интегрирования.

После прогона программы получаем следующие приближенные значения математического ожидания ошибки: (0,0289; 0,0187) - шаг равен 1/16; (0,0156; 0,0098) - шаг равен 1/32; (0,0073; 0,0050) - шаг равен 1/64.

Дисперсию (квадрат среднеквадратического отклонения) ошибки реализации можно подсчитать по формуле :

$$D[E] = M[E^2] - M^2[E].$$

После прогона программы получаем следующие значения дисперсии ошибки : (при сопоставлении вариантов реализации на CPU (FPU) одного и того же приближенного алгоритма): (0,00113; 0,00048) - шаг равен 1/16; (0,00031; 0,00013) - шаг равен 1/32; (0,00007; 0,00003) - шаг равен 1/64.

Прочие характеристики:

- максимальная глубина стека FPU равна шести;

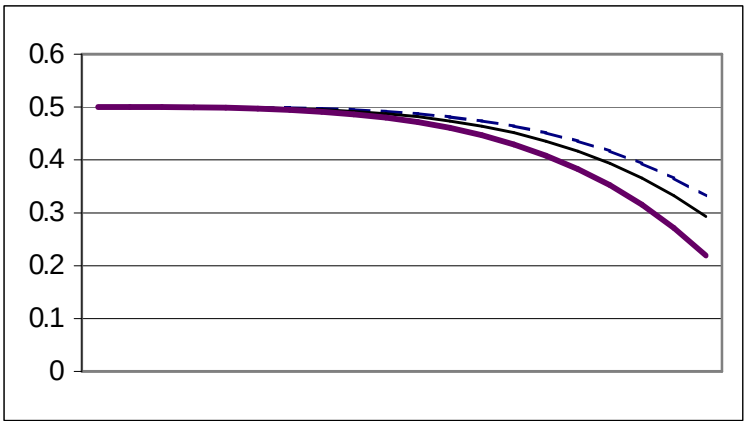

Рис. 1. Графики эталонного (аналитического) решения, результата интегрирования уравнения на языке ассемблера с использованием CPU и FPU. Сверху вниз: CPU; FPU; эталон. Шаг h = 1/16

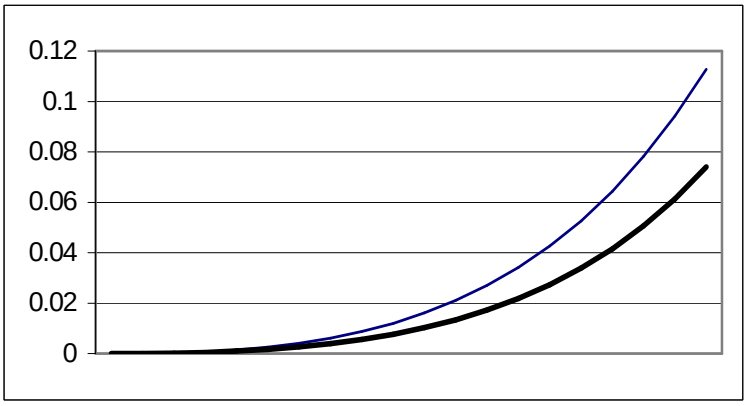

Рис. 2. Графики ошибки интегрирования (сверху вниз: "эталон-CPU" и "эталон-FPU"). Шаг h = 1/16

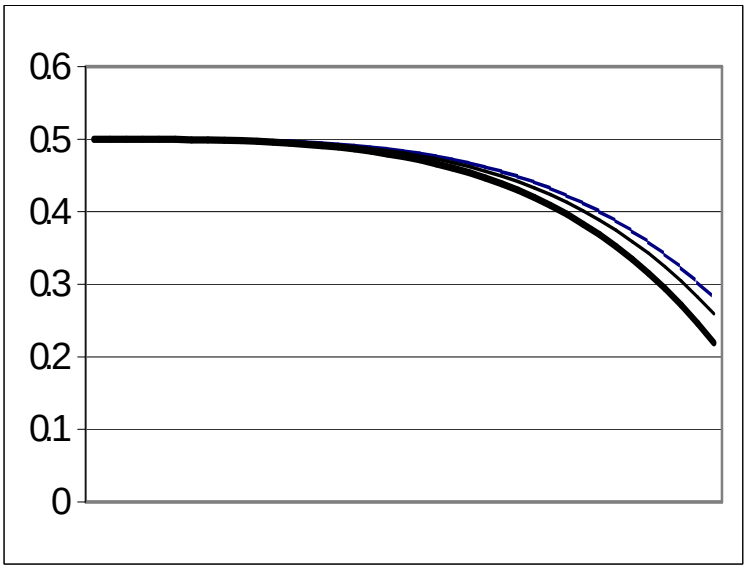

Рис. 3. Графики эталонного (аналитического) решения, результата интегрирования уравнения на языке ассемблера с использованием CPU и FPU. Сверху вниз: CPU; FPU; эталон. Шаг h = 1/32

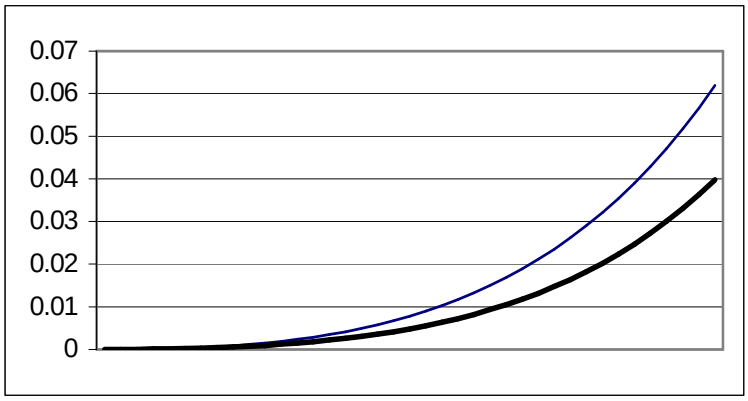

Рис. 4. Графики ошибки интегрирования (сверху вниз: "эталон-CPU" и "эталон-FPU"). Шаг h = 1/32

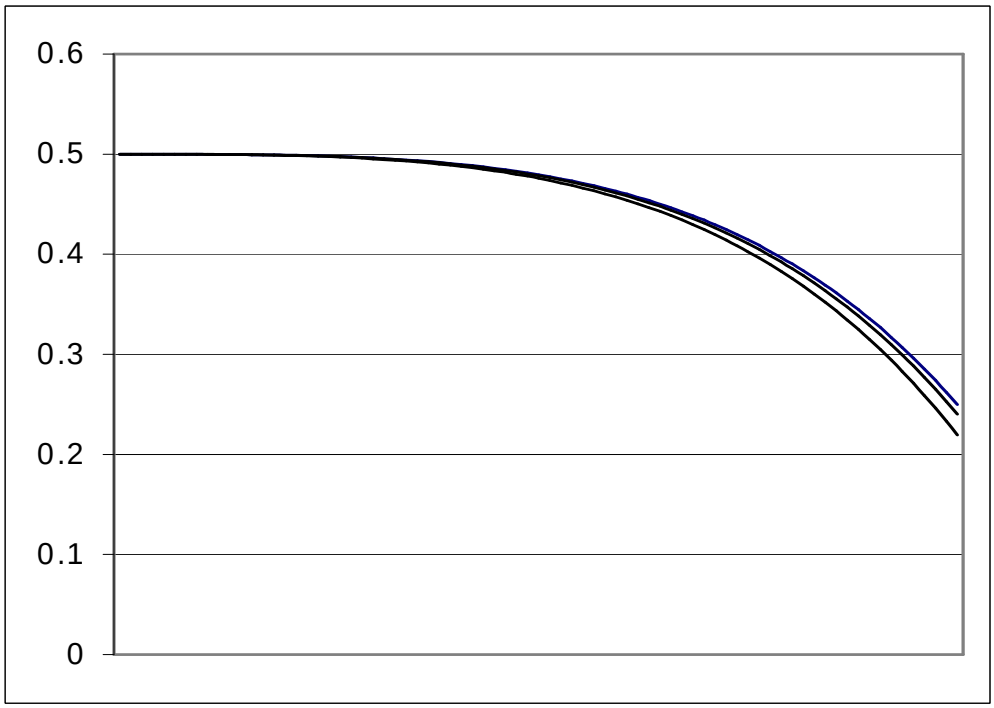

Рис. 5. Графики эталонного (аналитического) решения, результата интегрирования уравнения на языке ассемблера с использованием CPU и FPU. Сверху вниз: CPU; FPU; эталон. Шаг h = 1/64

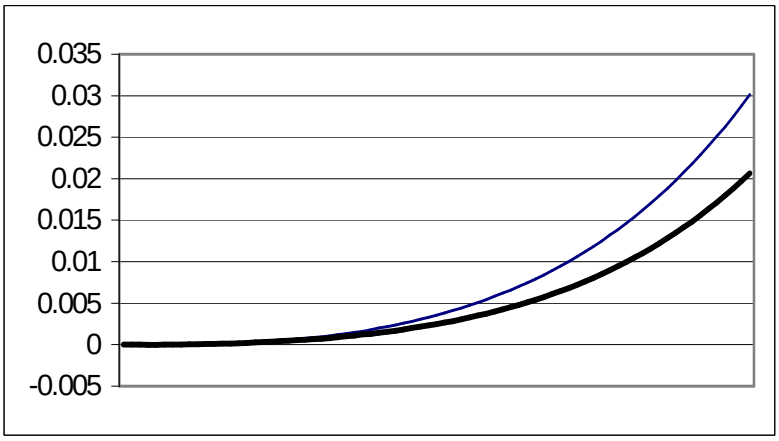

Рис. 6. Графики ошибки интегрирования (сверху вниз: "эталон-CPU" и "эталон-FPU").

Шаг h = 1/64

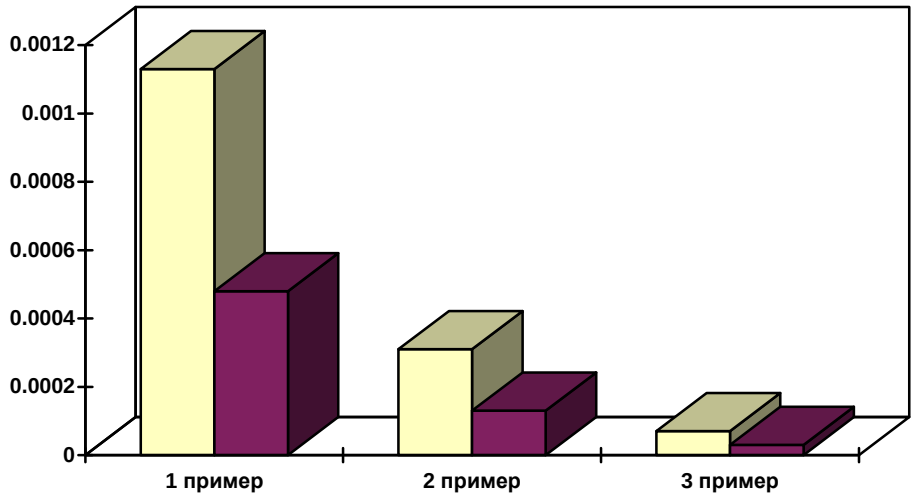

Рис. 7. Гистограммы значений дисперсии в зависимости от величины шага интегрирования

	1 пример	2 пример	3 пример
Величина шага h	0,0625	0,03125	0,015625
Число шагов n	19	38	76
Дисперсия D ошибки "CPU - эталон"	0,00113	0,00031	0,00007
Дисперсия D ошибки "FPU - эталон"	0,00048	0,00013	0,00003

Выводы :

Из гистограмм дисперсии ошибки (см. рис. 7) видно, что с уменьшением величины шага происходит повышение точности (дисперсия ошибки уменьшается), причем с уменьшением величины шага скорость изменения дисперсии ошибки уменьшается. Заметим, что вычисления на FPU выполнены точнее, чем на FPU (это видно из сравнения мат.ожидания и дисперсии ошибки для двух случаев). Убывание дисперсии ошибки при уменьшении шага происходит по закону, по форме похожему на показательный; интенсивность этого убывания сильнее для вычислений, выполненных на CPU.

Листинг программы

```
program Alex3pas;
Uses Graph;
const
  AlgScaleX = 16384;
  AlgScaleH = 8*32768;
  AlgScaleY = 32768;
  AlgScaleY1 = 32768;
  AlgScaleY2 = 8192;
  AlgScaleY3 = 4096;
  RightMrgPhysX = 1.2;
  StartCondY = 0.5;
  StartCondY1 = 0;
  StartCondY2 = 0;
  MaxRightIndex = 76;
  HPhys : extended = 1/64; {1/16, 1/32, 1/64}
var
   HP : byte;
   N, HMach, i, k : integer;
   CurCPURes, CurTemplateRes, CurFPURes : extended;
   CurError_CPU, CurError_FPU : extended;
   MachValsY_CPU : array[0..MaxRightIndex] of integer;
   InternValsY_FPU : array[0..MaxRightIndex] of extended;
   AverageError_CPU, AverageError_FPU,
   AverageSquaredError_CPU, AverageSquaredError_FPU,
   Disperse_CPU, Disperse_FPU : extended;
   TmpVal : extended;
   TF : text;
   S : string;

procedure AsmCalc(N,Y,Y1,Y2 : integer);
label Next, Finish;
begin
     asm
       mov     cx, N
       xor     si, si
       mov     ax, Y
       mov     word ptr ds : MachValsY_CPU [si], ax
       or      si, 2
       {}
       mov     ax, Y1
       mov     dx, HMach
Next:
       {z7(dx) = h*y1}
       imul      dx  {dx : ax}
       sar       dx, 1
       sar       dx, 1
       {z8 = z7(dx) + y}
       mov       ax, Y
       add       ax, dx
       {}
       mov       word ptr ds : MachValsY_CPU [si], ax
       {}
       inc     si
```

```
        inc     si
        dec     cx
        cmp     cx, 0
        jle     Finish
        {}
        mov     Y, ax
        {z2(dx) = y3 = z1 - y = y2 - y}
        mov     bx, y2
        push    bx
        {mov     ax, Y}
        {assumed that Y-old already in ax}
        neg     ax
        sar     ax, 1
        sar     ax, 1
        sar     ax, 1
        add     ax, bx  {y3}
        {mov     Y3, ax}
        {z3(dx) = h*y3}
        {mov     ax, HMach}
        mov     bx, HMach
        imul    bx  {dx : ax}
        sar     dx, 1
        {z4(dx) = z3(dx) + y2}
        pop     bx  {y2}
        add     dx, bx
        mov     Y2, dx
        {z5(dx) = h*y2(bx)}
        mov     ax, bx
        mov     dx, HMach
        push    dx
        imul    dx   {dx : ax}
        {z6(ax) = z5(dx) + y1}
        mov     ax, Y1
        push    ax
        add     ax, dx
        mov     Y1, ax
        {z7(dx) = h*y1}
        pop     ax {y1}
        pop     dx {h}
        jmp     Next
Finish:
    end;
end;

procedure FPUCalc(N,Y,Y1,Y2 : integer);
label Next,Finish;
var
   Mh,MY,MY1 : longint;
   MY2 : integer;
   const2 : integer;
   YPhys_internal,
   Y1Phys_internal,
   Y2Phys_internal,
   HPhys_internal,
   y3_internal   : extended;
{for conversion - Y,Y1,Y2,HMach}
```

```
begin
  Mh := 8*32768;
  MY := 32768;
  MY1 := 32768;
  MY2 := 8192;
  const2 := 2;
  asm
     finit
     xor        si,si
     {}
     fild       Y2
     fild       MY2
     fdivp       st(1),st  {res=st}
     fstp       Y2Phys_internal
     {}
     fild       HMach
     fild       MH
     fdivp       st(1),st  {res=st}
     fstp       HPhys_internal
     {}
     fild       Y1
     fild       MY1
     fdivp       st(1),st  {res=st}
     fst        st(1)
     fstp       Y1Phys_internal
     {}
     fild       Y
     fild       MY
     fdivp       st(1),st  {res=st}
     fst        st(1)
     fstp       YPhys_internal {y=st}
     {}
     mov         cx, N
Next:
      fst     st(1)
      fstp    extended ptr ds : InternValsY_FPU [si]
      dec     cx
      cmp     cx, 0
      jl      Finish
      add     si,10   {y=st}
      {z1(k)=2*y2(k)}
      fild      const2 {2,y}
      fld       Y2Phys_internal   {y2,2,y}
      fmulp     st(1), st {st=z1(k); y}
      {z2=y3=z1-y}
      fsub      st, st(1)  {st=z2}
      {z7=h*y1}
      fld       Y1Phys_internal
      fld       HPhys_internal  {h, y1}
      fmul      st, st(1) {st=z7; y1; y3}
      {z8 = z7+y}
      fld       YPhys_internal {y,z7,y1,y3}
      faddp      st(1), st  {st=z8, y1, y3}
      fstp       YPhys_internal  {y1, y3}
      {z5 = h*y2(k)}
      fld       HPhys_internal {H,y1, Y3}
```

```
        fst       st(3)        {H,y1,y3,h}
        fld       Y2Phys_internal  {Y2,h,y1,y3,h}
        fst       st(5)   {y2,h,y1,y3,h,y2}
        fmulp     st(1), st  {z5, y1, y3, h, y2}
        {z6 = z5 + y1}
        faddp     st(1), st  {z6, y3, h, y2}
        fstp      Y1Phys_internal {y3, h, y2}
        {z3 = h*y3}
        fmulp      st(1), st {z3, y2}
        {z4=z3+y2}
        faddp     st(1), st  {z4}
        fstp      Y2Phys_internal
        {}
        {we don't want overflow the FPU stack!}
        {fstp      st(0)}
        {}
        fld       YPhys_internal
        {}
        jmp       Next
Finish:
    end;
end;

function CalcTemplateY(X : extended) : extended;
const
 Lambda1 = 1;
 Lambda2 = 1.6180339887;
 Lambda3 = -0.6180339887;
 C1 = 0.5;
 C2 : extended = -0.223606797749979;
 C3 : extended = 0.223606797749979;
begin
  CalcTemplateY := C1*exp(Lambda1*X)+C2*exp(Lambda2*X)+C3*exp(Lambda3*X);
end;

begin
  writeln ('Введите значение, обратное шагу интегрирования.');
  writeln ('Например, если шаг h = 1/16, введите 16 :');
  readln(HP);
  HPhys := 1/HP;
  case HP of
   16:  assign(TF, 'C:\ALEX_WRK\MAXLABS\LAB2\lab2t_16.txt');
   32:  assign(TF, 'C:\ALEX_WRK\MAXLABS\LAB2\lab2t32.txt');
   64:  assign(TF, 'C:\ALEX_WRK\MAXLABS\LAB2\lab2t64.txt');
  end;
  HMach := trunc(AlgScaleH*HPhys);
  N := trunc(RightMrgPhysX/HPhys);
  AsmCalc(N, trunc(StartCondY*AlgScaleY),
        trunc(StartCondY1*AlgScaleY1),trunc(StartCondY2*AlgScaleY2));
  FPUCalc(N, trunc(StartCondY*AlgScaleY),
        trunc(StartCondY1*AlgScaleY1),trunc(StartCondY2*AlgScaleY2));
  AverageError_CPU := 0;
  Averageerror_FPU := 0;
  AverageSquaredError_CPU := 0;
  AverageSquarederror_FPU := 0;
  for i := 0 to N do
```

```
  begin
   CurCPURes := MachValsY_CPU[i]/AlgScaleY;
   CurFPURes := (trunc(InternValsY_FPU[i]*AlgScaleY))/AlgScaleY;
   CurTemplateRes := CalcTemplateY(i*HPhys);
   CurError_CPU := CurCPURes-CurTemplateRes;
   CurError_FPU := CurFPURes-CurTemplateRes;
   Averageerror_CPU := Averageerror_CPU + CurError_CPU;
   Averageerror_FPU := Averageerror_FPU + CurError_FPU;
   AverageSquaredError_FPU :=
     AverageSquaredError_FPU + CurError_FPU*CurError_FPU;
   AverageSquaredError_CPU :=
     AverageSquaredError_CPU + CurError_CPU*CurError_CPU;
  end;
  AverageError_CPU := (AverageError_CPU / (N+1));
  AverageError_FPU := (AverageError_FPU / (N+1));
  AverageSquaredError_CPU :=
    (AverageSquaredError_CPU / (N+1));
  AverageSquaredError_FPU :=
    (AverageSquaredError_FPU / (N+1));
  Disperse_CPU :=
    (AverageSquarederror_CPU - Averageerror_CPU*Averageerror_CPU);
  Disperse_FPU :=
    (AverageSquarederror_FPU - Averageerror_FPU*Averageerror_FPU);
  writeln('Математическое ожидание ошибки по CPU = ',
   AverageError_CPU : 15 : 15);
  writeln('Дисперсия ошибки по CPU = ',Disperse_CPU : 15 : 15);
  writeln('Математическое ожидание ошибки по FPU = ',
   AverageError_FPU : 15 : 15);
  writeln('Дисперсия ошибки по FPU = ',Disperse_FPU : 15 : 15);
  rewrite(TF);
  for i := 0 to N do
  begin
    TmpVal := MachValsY_CPU[i]/AlgScaleY;
    Str(TmpVal:2:15, S);
    write(TF,S);
    write(TF,' ');
    TmpVal := (trunc(InternValsY_FPU[i]*AlgScaleY))/AlgScaleY;
    Str(TmpVal:2:15, S);
    write(TF,S);
    write(TF,' ');
    TmpVal := CalcTemplateY(i*HPhys);
    Str(TmpVal:2:15, S);
    writeln(TF,S);
  end;
  close(TF);
  readln;
end.
```