НИУ ВШЭ
Факультет Компьютерных Наук
Сделано Коптевым Олегом
Станиславовичем
Из группы БПИ197

Программирование на языке ассемблера. Микропроект. 2020-2021 уч.г.

Программа вычисления корня кубического из заданного числа согласно быстросходящемуся итерационному алгоритму определения корня n-ной степени с точностью не хуже 0,05%

Оглавление

1. Алгоритм

Для нахождения кубического корня используется [алгоритм нахождения корня n-ной степени].

Для определения точности вычисления используется такая проверка:

$$\frac{100 \cdot |x_k - x_{k-1}|}{x_k} < 0.05$$

1.1 Суть алгоритма

1) Сделать предположение x_0;

2) Задать $x_{k+1} = \frac{1}{n}\left((n-1)x_k + \frac{A}{x_k^{n-1}}\right)$, где в нашем случае $n = 3$;

3) Повторять шаг 2, пока не будет достигнута необходимая точность.

2. Компиляция и сборка

Компилируется при помощи вызова `fasm app.asm`, сборка - `gcc app.o -o app -lm` (флаг `-lm` нужен, чтобы использовать библиотеку `math.h`)

3. Пример работы

Ниже представлены несколько вариантов входных данных и соответствующие результаты исполнения:

Исходное число: 8, результат: 2.000000

```
parallels@debian:/media/psf/CubeRoot$ ./app
Program for counting cube root with the precision of 0.05%
Please enter the number: 8
Listing approximations...
        1: 2.666667
        2: 2.152778
        3: 2.010586
        4: 2.000056
Calculated result: 2.000000
True result using cbrt(): 2.000000
parallels@debian:/media/psf/CubeRoot$
```

Исходное число: 970299, результат: 99.000000

```
parallels@debian:/media/psf/CubeRoot$ ./app
Program for counting cube root with the precision of 0.05%
Please enter the number: 970299
Listing approximations...
        1: 323433.000000
        2: 215622.000003
        3: 143748.000009
        4: 95832.000022
        5: 63888.000050
        6: 42592.000112
        7: 28394.666920
        8: 18929.778348
        9: 12619.853134
        10: 8413.237454
        11: 5608.829539
        12: 3739.229973
        13: 2492.843115
        14: 1661.947457
        15: 1108.082069
        16: 738.984794
        17: 493.248791
        18: 330.161917
        19: 223.075032
        20: 155.216225
        21: 116.902353
        22: 101.601609
        23: 99.066048
        24: 99.000044
Calculated result: 99.000000
True result using cbrt(): 99.000000
parallels@debian:/media/psf/CubeRoot$
```

Исходное число: -27, результат: -3.000000

```
parallels@debian:/media/psf/CubeRoot$ ./app
Program for counting cube root with the precision of 0.05%
Please enter the number: -27
Listing approximations...
        1: -9.000000
        2: -6.111111
        3: -4.315066
        4: -3.360067
        5: -3.037207
        6: -3.000454
Calculated result: -3.000000
True result using cbrt(): -3.000000
parallels@debian:/media/psf/CubeRoot$
```

Исходное число: 0, результат: 0

```
parallels@debian:/media/psf/CubeRoot$ ./app
Program for counting cube root with the precision of 0.05%
Please enter the number: 0
Listing approximations...
        1: 0.000000
Calculated result: 0.000000
True result using cbrt(): 0.000000
parallels@debian:/media/psf/CubeRoot$
```

Исходное число: 99999999999999, результат: 46415.888340

```
parallels@debian:/media/psf/CubeRoot$ ./app
Program for counting cube root with the precision of 0.05%
Please enter the number: 99999999999999
Listing approximations...
        1: 33333333333333.000000
        2: 22222222222222.000000
        3: 14814814814814.666016
        4: 9876543209876.443359
        5: 6584362139917.628906
        6: 4389574759945.085938
        7: 2926383173296.724121
        49: 119137.337611
        50: 81773.350641
        51: 59500.452268
        52: 49082.355961
        53: 46558.124686
        54: 46416.322430
Calculated result: 46415.888340
True result using cbrt(): 46415.888336
parallels@debian:/media/psf/CubeRoot$
```

Во всех случаях вычисленное значение лежит в области допустимой погрешности

Приложение 1. Исходный код с комментариями

```
format ELF64
public main

; Koptev Oleg
; Variant 10

extrn printf
extrn scanf
extrn cbrt

section '.data' writable
    strStart        db  'Program for counting cube root with the precision of %.2f%', 10, 0
    strInFloat      db  'Please enter the number: ', 0
    strScanFloat    db  '%lf', 0
    strCalcRes      db  'Calculated result: %f', 10, 0
    strTrueRes      db  'True result using cbrt(): %f', 10, 0
    strFiller       db  'Listing approximations...', 10, 0

    strIterPhrase   db  9, '%d: %f', 10, 0

    A               dq  ?
    tempA           dq  ?
    epsilon         dq  0.05
    x               dq  ?
    tempX           dq  ?
    input_A         dq  ?
    next_x          dq  ?
    true_val        dq  ?
    counter         dd  0
```

```
    zero            dd  0
    two             dd  2
    three           dd  3
    hundred         dd  100

section '.code' writable

; We start with several phrases for a user to understand what the program does
; Then we get a double number from user
; Calling the function to process the input information
; That function makes first assumption: x / 3
; Then it counts the next member of the sequence and checks if its precision is good enough
; In case the difference is bigger than 0.05% we repeat the previous step
; In the other case we finish the processing step and just output the calculated number

main:
    ; printf("Program for counting cube root with the precision of %.2f\n", epsilon);
    sub rsp, 40
    mov rdi, strStart
    movsd xmm0, [epsilon]
    mov eax, 1
    call printf

    ; printf("Please enter the number: ");
    mov rdi, strInFloat
    xor rax, rax
    call printf

    ; scanf("%lf", &A);
```

```
    mov rdi, strScanFloat
    mov rsi, A
    call scanf

    ; input_A = A;
    mov rdx, [A]
    mov [input_A], rdx

    ; start();
    call start

    ; printf("Calculated result: %.3f\n", next_x);
    mov rdi, strCalcRes
    movsd xmm0, [next_x]
    mov eax, 1
    call printf

    ; cbrt(input_A);
    movsd xmm0, [input_A]
    mov eax, 1
    call cbrt

    ; printf("True result using cbrt(): %.3f\n");
    mov rdi, strTrueRes
    mov eax, 1
    call printf

    add rsp, 40
    ; return 0;
    xor rax, rax
    ret
```

```
; void start()
;   input: user number is A variable
;   output: cube root in x variable
start:
    ; printf("Listing approximations...\n");
    mov rdi, strFiller
    xor rax, rax
    call printf

    ; next_x = A / 3;
    fld qword [A]
    fild dword [three]
    fdivp st1, st0
    fstp qword [next_x]

    ; counter = 1;
    mov ebx, 1
    mov [counter], ebx

    xor rcx, rcx

    ; do
    .iter:
        ; printf("%d: %f", counter, next_x)
        sub rsp, 40
        mov rdi, strIterPhrase
        mov esi, [counter]
        movsd xmm0, [next_x]
        mov eax, 1
        call printf
        add rsp, 40

        ; x = next_x;
```

```
        fld qword [next_x]
        fstp qword [x]

        ; next_x = getNext()
        call getNext
        movsd qword [next_x], xmm0

        ; if (next_x == null) goto reset_x;
        fld qword [next_x]
        fcomi st1
        jp reset_x
        fstp st0

        ; while (abs(next_x - x) * 100 / next_x > epsilon)
        fld qword [epsilon]
        fld qword [next_x]
        fld qword [x]
        fsubp st1, st0
        fild dword [hundred]
        fmulp st1, st0
        fld qword [next_x]
        fdivp st1, st0
        fabs

        fcomi st1
        jb exit

        ; empty the fpu stack to avoid stack overflow
        fstp st0
        fstp st0
        fstp st0

        ; counter++;
```

```
        mov edx, [counter]
        inc edx
        mov [counter], edx

        jmp .iter
reset_x:
    fild dword [zero]
    fstp qword [next_x]
exit:
    ret

; double getNext(double A, double x)
;   input: user number, current approximation
;   output: (2 * x + A / (x * x)) / 3
getNext:
    ; tempA = A / (x * x);
    fld qword [A]
    fld qword [x]
    fmul st0, st0
    fdivp st1, st0
    fstp qword [tempA]

    ; tempX = 2 * x;
    fld qword [x]
    fild dword [two]
    fmulp st1, st0
    fstp qword [tempX]

    ; tempA = tempA + tempX;
    fld qword [tempA]
    fld qword [tempX]
    fadd st0, st1
```

```
    ; tempA = tempA / 3;
    fild dword [three]
    fdivp st1, st0
    fstp qword [tempA]

    movsd xmm0, qword [tempA]
    ret
```