МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №5

# по дисциплине: Архитектура вычислительных систем тема:

«Команды сопроцессора»

# Выполнил: ст. группы ПВ-223

# Дмитриев Андрей Александрович

Проверил:

# Осипов Олег Васильевич

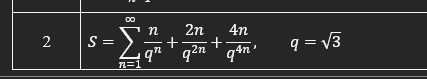
Белгород 2024 г.

# Вариант 2

Цель работы: изучение команд сопроцессора для выполнения арифметических операций

Задания для выполнения к работе

1. Написать функцию pow (*x*, *y*) для возведения числа *x* в степень *y*. Числа *x*, *y* могут быть произвольными, в том числе отрицательными. Рассмотреть случаи, когда *x* = 0 и/или *y* = 0. Аргументы передавать подпрограмме через стек. Если алгоритм требует выгрузки чисел из сопроцессора в память или регистры, использовать для этого стек. Подобрать набор тестовых данных для проверки работы функции pow (не менее 10). Убедиться в том, что результаты работы написанной функции pow и стандартной функции pow библиотеки math.h языка C или функции Math.Pow языка C# совпадают. В отчёт включить текст программы, блок-схему алгоритма функции pow и набор тестовых данных.
2. Численно исследовать на сходимость ряд. Аргументы тригонометрических функций считать в радианах. Для возведения чисел в степень использовать написанную функцию pow. В отчёт включить текст программы и значения суммы ряда при *n* от 1 до 50. Вывести результат на экран в виде: n = 1; S = … n = 2; S = …



# Задание №1.

Для возведения числа 2 в произвольную степень также нет команды. Команда **F2XM1** корректно считает только тогда, когда в ST(0) находиться число в интервале (-1, 1). Поэтому число *t* = *y*log2*x* нужно разбить на целую (*a*) и дробную части (*b*). Дробную часть можно получить в результате вычисления командой **FPREM** частичного остатка от деления *t* на единицу. Есть другой способ. Если сначала получить целую часть *a* с помощью округления до ближайшего меньшего целого, то *b* = *t* - *a*. Тогда 2𝑡 = 2𝑎+𝑏 = 2𝑎2𝑏.

2*a* нужно вычислять помощью команды **FSCALE**. Целую часть можно специально не вычислять, так как команда **FSCALE** сама округляет число в ST(1) до ближайшего меньшего целого. 2*b* теперь можно вычислить командой **F2XM1**, но после выполнения этой команды к результату нужно прибавить единицу. Если *x* < 0, то нужно просто вынести знак: (-*x*)*y* = *xy*. Если *y* < 0, то *x*-*y* = 1 / *xy*.

Код:

; x ^ y = 2 ^ (y \* logx)

; 2 ^ t

; t = y log x

; дробная часть = b

; целая часть = a( с округлением до целого меньшего)

; b = t - a

; 2 ^ a \* 2 ^ b

; результат записывается в edx:eax

pow proc x\_: real8, y\_: real8

sub esp, 108    ; для сохранения состояния сопроцессора

fsave [esp] ; сохраняем состояние сопроцессора

; задаём значение управляещего регистра CWR

; такое, чтобы округление происходило

; в меньшую сторону

sub esp, 2  ; память под значение регистра CWR xor eax, eax

fstcw [esp] ; сохраняем значение в память

mov ax, [esp]

and ax, 0f0ffh

or ax, 0600h

mov [esp], ax

fldcw [esp]

add esp, 2

xor eax, eax

xor edx, edx

; начало вычислений

fld y\_

fldz

db 0dfh, 0f0h+1 ;FCOMIP y, 0 (сравнение y и 0)

jne y\_none\_zero ;y = 0 -> выход st0 = 1

y\_zero:

fld1

jmp copy\_top\_to\_result

y\_none\_zero:

fld x\_

; сравнение с нулём

fldz

db 0dfh, 0f0h+1

jne start\_calculating

fldz

jmp copy\_top\_to\_result

start\_calculating:

jb non\_negative

inc edx

fabs

non\_negative:

; ST = x, y

fxch ST(1)

; st = y, x

fld ST(0)

; st = |\_ y \_|, y, x

frndint

db 0dfh, 0f0h+1

jne is\_reminder\_exist

no\_reminder:

sub ESP, 4

fist dword ptr [ESP]

mov EAX, [ESP]

ADD ESP, 4

AND EAX, 1

is\_reminder\_exist:

FXCH ST(1)

fyl2x   ; ST(0) = ylogx

fld ST(0)   ; ST(0) = ylogx

frndint ; ST = a, t

FSUB ST(1), ST(0)   ; ST(1) = b; st = a, b

fld1    ; ST(0) = 1

fscale  ; ST(0) = 2^a   ; ST(0) = 2^a, ST(1) = a, ST(2) = t - a = b

fld ST(2)   ; ST(0) = b, ST(1) = 2^a, ST(2) = a

f2xm1

fld1

faddp ST(1), ST(0)

fmulp ST(1), ST(0)  ; результат в вершине стека

cmp eax, 0

je copy\_top\_to\_result

cmp edx, 0

je copy\_top\_to\_result

fchs

copy\_top\_to\_result:

sub esp, 8

fstp real8 ptr [esp]

mov eax, [esp]

mov edx, [esp + 4]

add esp, 8

load\_state:

frstor [esp]

add esp, 108

ret

pow endp

START:

    push dword ptr y[4]

    push dword ptr y[0]

    push dword ptr x[4]

    push dword ptr x[0]

    call pow

    ; Вывод результата в консоль

    push edx

    push eax

    push  offset format

    call  crt\_printf

    ; Завершаем программу

    push 0

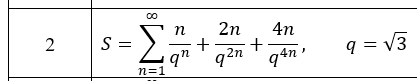
    call ExitProcess

END START

Тестовые данные:

|  |  |  |
| --- | --- | --- |
| x | y | res |
| 2.0 | 3.0 | 8.0 |
| -2.0 | 3.0 | -8.0 |
| 2.0 | -3.0 | 0.125 |
| -2.0 | -3.0 | -0.125 |
| 16.0 | 0.5 | 4.0 |

Задание №2.



; x ^ y = 2 ^ (y \* logx)

; 2 ^ t

; t = y log x

; дробная часть = b

; целая часть = a( с округлением до целого меньшего)

; b = t - a

; 2 ^ a \* 2 ^ b

; результат записывается в edx:eax pow proc x\_: real8, y\_: real8

sub esp, 108 ; для сохранения состояния сопроцессора fsave [esp] ; сохраняем состояние сопроцессора

; задаём значение управляещего регистра CWR

; такое, чтобы округление происходило

; в меньшую сторону

sub esp, 2 ; память под значение регистра CWR xor eax, eax

fstcw [esp] ; сохраняем значение в память mov ax, [esp]

and ax, 0f0ffh or ax, 0600h mov [esp], ax fldcw [esp] add esp, 2

xor eax, eax xor edx, edx

; начало вычислений fld y\_

fldz

db 0dfh, 0f0h+1 ;FCOMIP y, 0 (сравнение y и0) jne y\_none\_zero ;y = 0 -> выход st0 = 1

y\_zero:

fld1

jmp copy\_top\_to\_result y\_none\_zero:

fld x\_

; сравнение с нулём fldz

db 0dfh, 0f0h+1

jne start\_calculating

fldz

jmp copy\_top\_to\_result

start\_calculating:

jb on\_negative inc edx

fabs non\_negative:

; ST = x, y

fxch ST(1)

; st = y, x fld ST(0)

; st = |\_ y \_|, y, x frndint

db 0dfh, 0f0h+1

jne is\_reminder\_exist no\_reminder:

sub ESP, 4

fist dword ptr [ESP] mov EAX, [ESP]

ADD ESP, 4

AND EAX, 1

is\_reminder\_exist:

FXCH ST(1)

fyl2x ; ST(0) = ylogx

fld ST(0) ; ST(0) = ylogx

frndint ; ST = a, t

FSUB ST(1), ST(0) ; ST(1) = b; st = a, b

fld1 ; ST(0) = 1

fscale ; ST(0) = 2^a ; ST(0) = 2^a, ST(1)

= a, ST(2) = t - a = b

fld ST(2) ; ST(0) = b, ST(1) = 2^a, ST(2) = a

f2xm1 fld1

faddp ST(1), ST(0)

fmulp ST(1), ST(0) ; результат в вершине стека

cmp eax, 0

je copy\_top\_to\_result

cmp edx, 0

je copy\_top\_to\_result

fchs copy\_top\_to\_result:

sub esp, 8

fstp real8 ptr [esp] mov eax, [esp]

mov edx, [esp + 4]

add esp, 8

load\_state:

frstor [esp] ; загружаем состояниесопроцессора add esp, 108

ret pow endp

move\_to\_esp\_real8 macro val\_ sub esp, TYPE real8

fld real8 ptr val\_ fstp real8 ptr [esp]

endm

move\_top\_to\_esp macro sub esp, TYPE real8

fstp real8 ptr [esp] endm

copy\_result\_to macro p\_ mov [p\_], eax

mov [p\_ + 4], edx endm

calculate\_sum\_of\_series proc uses ecx n\_ : dword LOCAL sum\_ : real8

LOCAL current\_grade\_q\_ : real8

sub esp, 108 ; для сохранения состояния сопроцессора

fsave [esp] ; сохраняем состояние сопроцессора

fldz

fst current\_grade\_q\_ ; обнуляем локальные переменные fstp sum\_

fld1 ; st = n1

mov ecx, n\_ calculate\_sum:

fld st(0) fld st(0)

; st = n, n

; посчитаем q \*\* n

; fst real8 ptr currentValues[0]

move\_top\_to\_esp move\_to\_esp\_real8 q

call pow ; результат -> edx:eax

; copy\_result\_to current\_grade\_q\_ mov dword ptr[current\_grade\_q\_], eax

mov dword ptr[current\_grade\_q\_ + 4], edx

fld current\_grade\_q\_ ; st = q\*\*n, n

fld st(0) ; st = q\*\*n, q\*\*n, n fst real8 ptr currentValues[1 \* TYPE real8]

fdivr st(0), st(2) ; st = n/q\*\*n , q\*\*n, n

fadd sum\_ ; st = sum + cur\_sum

fstp sum\_ ; sum = cur\_sum

; st = q \*\* n, n

fmul st(0), st(0) ; q\*\*2n

fxch st(1) ; n, q\*\*2n

fld1

fadd st(0), st(0) ; st = 2, n, q\*\*2n

fmulp st(1), st(0) ; st = 2n, q\*\*2n

; fst real8 ptr currentValues[3 \* TYPE real8] fxch st(1) ; q\*\*2n, 2n

fst real8 ptr currentValues[2 \* TYPE real8]

fld st(0) ; q\*\*2n, q\*\*2n, 2n

fdivr st(0), st(2) ; st = 2n/q\*\*2n , q\*\*2n, 2n

; fst real8 ptr currentValues[5 \* TYPE real8]

fadd sum\_ ; st = sum + cur\_sum

fstp sum\_ ; sum = cur\_sum

; st = q \*\* n, n

fmul st(0), st(0) ; q\*\*4n

fst real8 ptr currentValues[3 \* TYPE real8] fxch st(1) ; n, q\*\*4n

fld1

fadd st(0), st(0) ; st = 2, n, q\*\*4n

fmulp st(1), st(0) ; st = 4n, q\*\*4n

; fst real8 ptr currentValues[6 \* TYPE real8]

fxch st(1) ; q\*\*4n, 4n

fld st(0) ; q\*\*4n, q\*\*4n, 4n

fdivr st(0), st(2) ; st = n/q\*\*n , q\*\*n, n

; fst real8 ptr currentValues[8 \* TYPE real8]

fadd sum\_ ; st = sum + cur\_sum, q\*\*n, n

fstp sum\_ ; st = q\*\*n, n

fstp st(0) ; n

fstp st(0) fld1

faddp st(1), st(0) dec ecx

cmp ecx, 0

jne calculate\_sum

load\_state:

frstor [esp] ; загружаем состояние сопроцессора add esp, 108

mov eax, dword ptr [sum\_]

mov edx, dword ptr [sum\_ + 4]

fst real8 ptr currentValues[0] fld sum\_

ret

calculate\_sum\_of\_series endp

main proc

mov ecx, 1 print\_sums\_for:

cmp ecx, 50

jg print\_sums\_for\_end

mov cur\_n, ecx

invoke calculate\_sum\_of\_series, cur\_n

mov ebx, offset currentValues mov dword ptr [ebx], eax

mov dword ptr [ebx + 4], edx

sub esp, sizeof currentValues mov esi, offset currentValues

mov edi, esp mov ecx, 4 start:

mov eax, dword ptr [esi] mov [edi], eax

mov eax, dword ptr [esi + 4] mov [edi + 4], eax

add esi, TYPE real8 add edi, TYPE real8

loop start

; sub esp, TYPE real8

; mov ebx, offset cur\_sum

; mov eax, [ebx]

; mov dword ptr [esp], eax

; mov eax, [ebx + 4]

; mov dword ptr [esp + 4], eax

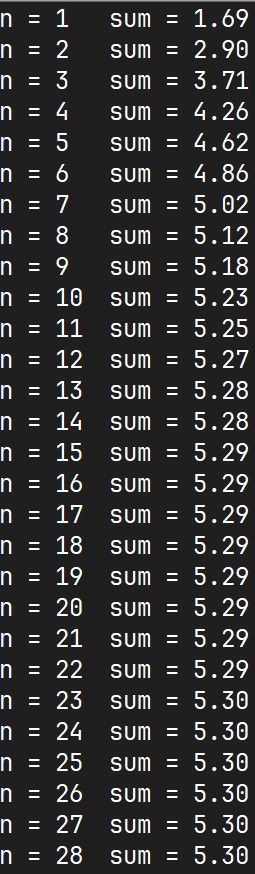
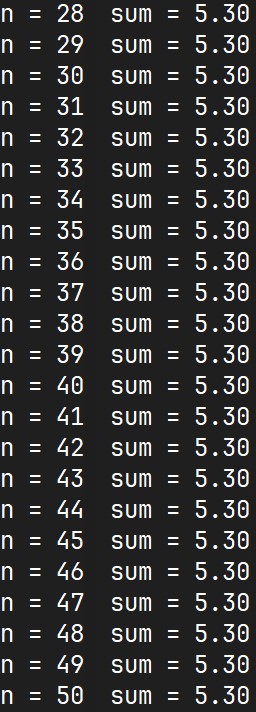
invoke crt\_printf, offset resultFmt, cur\_n add esp, sizeof currentValues

mov ecx, cur\_n inc ecx

jmp print\_sums\_for print\_sums\_for\_end:

invoke ExitProcess, 0 main endp

END

Вывод: изучил команды сопроцессора для выполнения арифметических операций