Учреждение Образования

«Белорусский Государственный Университет Информатики и Радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра Информатики

Дисциплина: АВС

Отчет

По лабораторным работам № 4-6

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ С ЧИСЛАМИ

С ПЛАВАЮЩЕЙ ТОЧКОЙ

Выполнил студент

Группы 253505

Волчецкий А.М.

Минск 2014

### *Задание к лабораторной работе 4*

Написать программу эмулятора АЛУ, реализующего *операции сложения и вычитания с плавающей* точкой над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

.MODEL SMALL

.386

STACK 100h

.STACK

.DATA

inputFirstNumberMessage db 10,13,'Print first number: $'

inputSecondNumberMessage db 10,13,'Print second number: $'

resultOfSumMessage db 10,13,'Sum: $'

resultOfDifferenceMessage db 10,13,'Difference: $'

choiceMessage db 10,13,'1-Sum 2-Difference: $'

pressAnyKeyMessage db 10,13,'Press any key...$'

owerflowMessage db 'Overflow$'

negativ dd 0

minus db 0

firstNumber dd ?

secondNumber dd ?

.CODE

; макрос выводит на экран строку

OutputString macro string

push ax

mov dx, offset string

mov ah, 9

int 21h

pop ax

endm

start:

mov ax, @data

mov ds, ax

OutputString inputFirstNumberMessage ;вводим первое число

call InputNumber

fstp firstNumber

OutputString inputSecondNumberMessage ;вводим второе число

call InputNumber

fstp secondNumber

OutputString choiceMessage ; выбираем--или суммировать или отнимать

choiceMetka:

call InputDigit

cmp al, 1

je sumMetka

cmp al, 2

je differenceMetka

jmp choiceMetka

sumMetka:

OutputString resultOfSumMessage ; суммируем их

mov eax, firstNumber

mov ebx, secondNumber

call Sum

call outfloat

jmp exit

differenceMetka:

OutputString resultOfDifferenceMessage

mov eax, firstNumber ; отнимаем их

mov ebx, secondNumber

mov minus,1

call Sum

call outfloat

jmp exit

\_oveflow:

OutputString owerflowMessage

jmp exit

;входные значения и результат в ecx

GetMantissa proc

and ecx, 8388607 ;все, что не мантисса станет нулем

or ecx, 8388608 ; устанавливаем в старшем разряде мантиссы 1

shl ecx, 6

ret

GetMantissa endp

; получает показатели числа и проверяет их на переполнение

GetExponent proc

shl ecx, 1

shr ecx, 24

cmp ecx, 255

jae \_oveflow

ret

GetExponent endp

;результат в eax

Sum proc

push ebx

push ecx

push edx

push edi

push esi

;сохраняем операнды

mov edi, eax

mov esi, ebx

push eax

push ebx

; получаем мантиссы

; в eax хранится мантисса firstNumber

; в ebx хранится мантисса secondNumber

mov ecx, eax

call GetMantissa

mov eax, ecx

mov ecx, ebx

call GetMantissa

mov ebx, ecx

;получаем порядки чисел

;в edi порядок firstNumber

;в esi порядок secondNumber

mov ecx, edi

call GetExponent

mov edi, ecx

mov ecx, esi

call GetExponent

mov esi, ecx

;восстанавливаем изначальные операнды

pop edx

pop ecx

; порядок firstNumber больше secondNumber--меняем местами слагаемые

cmp edi, esi

jle exponentEqual

xchg edi, esi ;меняем порядок

xchg eax, ebx ;меняем манциссы

mov negativ,1

; Выравниваем порядки

exponentEqual:

cmp edi, esi

je checkMantissasSign

inc edi ; здесь так сказать делим на 10 => порядок увеличивается на 1, а в мантиссе точка переносится влево

shr eax, 1

jmp exponentEqual

checkMantissasSign:

test ecx, ecx

jns firstNumberIsPositive ; jns=> если знак плюс

neg eax

firstNumberIsPositive:

test edx, edx

jns secondNumberIsPositive

neg ebx

secondNumberIsPositive:

cmp minus,1 ; если minus=1, значит мы числа вычитаем

jne plus

sub eax, ebx

jmp checkMantissaSign

plus:

add eax, ebx

checkMantissaSign:

mov esi, 0

test eax, eax

jz endSum ; если в мантиссе 0, то выполнять дальнейшие действия нету смысла(Ответ: 0)

jns mantissaIsPositiv

neg eax

cmp negativ,1

je mantissaIsPositiv

mov esi, 80000000h ; результат отрицательный

mantissaIsPositiv:

; в edi порядок находится

inc edi

normalizationOfMantissa:

bt eax, 30 ; значение 30-го бита перейдет во флаг CF

jc orderCheckOwerflow ; если CF=1 то переходим

shl eax, 1

dec edi

jmp normalizationOfMantissa

orderCheckOwerflow:

cmp edi, 255

jl makeResult

jmp \_oveflow

;все засовываем в один регистр -- eax

makeResult:

shl edi, 23 ; порядок будет находится в 8 старших битах

shl eax, 2 ; делаем, чтобы самый старший разряд начинался с мантиссы

shr eax, 9 ; возвращаем мантиссу на свое место

or esi, edi ; прибавляем порядок

or esi, eax ; прибавляем мантиссу

mov eax, esi ; результат в eax

endSum:

pop esi

pop edi

pop edx

pop ecx

pop ebx

ret

Sum endp

OutputSymbol macro simbol

push ax

push dx

mov dl, simbol

mov ah, 02h

int 21h ; выводит на экран содержимое регистра dl. т.е то, что мы ввели

pop dx

pop ax

endm

InputDigit proc

startInputSymbol:

mov ah, 08h

int 21h

cmp al, 13

je endInputDigit

cmp al, '.'

je printSymbol

cmp al, '-'

je printSymbol

cmp al, 39h

ja startInputSymbol

cmp al, 30h

jb startInputSymbol

symbolToDigit:

OutputSymbol al

sub al, 30h

jmp endInputDigit

printSymbol:

OutputSymbol al

endInputDigit:

ret

InputDigit endp

InputNumber proc

push ax

push dx

push si

mov si, 0 ; если в si хранится 0--значит число положительное

push bp

mov bp, sp

push 10

push 0

fldz ; добавляем 0 в стек сопроцессора

call InputDigit

cmp al, '-'

jne numberIsPositive

mov si, 1 ; значит вводимое число отрицательное

inputIntegerPart:

call InputDigit

numberIsPositive:

cmp al, 13

je checkSign

cmp al, '.'

je pointInputed

mov [bp - 4], al ;сохраняем введенную цифру во временной ячейке

fimul word ptr [bp - 2] ; множим существующее число на 10

fiadd word ptr [bp - 4] ; прибавляем к нему сохраненную ранее цифру

jmp inputIntegerPart

pointInputed:

fld1 ;добавляем в стек сопроцессора 1

inputFractionalPart:

call InputDigit

cmp al, 13

je endInputFractionalPart

mov [bp - 4], al ;сохраняем введенную цифру во временной ячейке

fidiv word ptr [bp - 2] ; получаем отрицательную степень десятки

fld st(0) ; продублируем ее

fimul word ptr [bp - 4] ; умножаем степень на введенную цифру

faddp st(2), st ; добавляем к текущему результату

jmp inputFractionalPart

endInputFractionalPart:

fstp st(0) ; выталкиваем степень десятки из стека

checkSign: ;здесь учитываем знак вводимого числа

test si, si

jz endInputNumber

fchs

endInputNumber:

leave ; восстанавливаем значение bp и sp, что было в начале

pop si

pop dx

pop ax

ret

InputNumber endp

outfloat proc near

push ax

push cx

push dx

outNumber dd 0

mov outNumber, eax

fld outNumber

; Формируем кадр стэка, чтобы хранить в нём десятку

push bp

mov bp, sp

push 10

push 0

; Проверяем число на знак, и если оно отрицательное,

ftst

fstsw ax

sahf

jnc @of1

; то выводим минус

mov ah, 02h

mov dl, '-'

int 21h

; и оставляем модуль числа.

fchs

; Пояснение далее пойдёт на примере. ; ST(0) ST(1) ST(2) ST(3) ...

; Отделим целую часть от дробной. ; 73.25 ... что-то не наше

@of1: fld1 ; 1 73.25 ...

fld st(1) ; 73.25 1 73.25 ...

; Остаток от деления на единицу даст дробную часть.

fprem ; 0.25 1 73.25 ...

; Если вычесть её из исходного числа, получится целая часть.

fsub st(2), st ; 0.25 1 73 ...

fxch st(2) ; 73 1 0.25 ...

; Сначала поработаем с целой частью. Считать количество цифр будем в CX.

xor cx, cx

; Поделим целую часть на десять,

@of2: fidiv word ptr [bp - 2] ; 7.3 1 0.25 ...

fxch st(1) ; 1 7.3 0.25 ...

fld st(1) ; 7.3 1 7.3 0.25 ...

; отделим дробную часть - очередную справа цифру целой части исходного числа,-

fprem ; 0.3 1 7.3 0.25 ...

; от чатсного оставим только целую часть

fsub st(2), st ; 0.3 1 7 0.25 ...

; и сохраним цифру

fimul word ptr [bp - 2] ; 3 1 7 0.25 ...

fistp word ptr [bp - 4] ; 1 7 0.25 ...

inc cx

; в стэке.

push word ptr [bp - 4]

fxch st(1) ; 7 1 0.25 ...

; Так будем повторять, пока от целой части не останется ноль.

ftst

fstsw ax

sahf

jnz short @of2

; Теперь выведем её.

mov ah, 02h

@of3: pop dx

; Вытаскиваем очередную цифру, переводим её в символ и выводим.

add dl, 30h

int 21h

; И так, пока не выведем все цифры.

loop @of3 ; 0 1 0.25 ...

; Итак, теперь возьмёмся за дробную часть, для начала проверив её существование.

fstp st(0) ; 1 0.25 ...

fxch st(1) ; 0.25 1 ...

ftst

fstsw ax

sahf

jz short @of5

; Если она всё-таки ненулевая, выведем точку

mov ah, 02h

mov dl, '.'

int 21h

; и не более шести цифр дробной части.

mov cx, 6

; Помножим дрообную часть на десять,

@of4: fimul word ptr [bp - 2] ; 2.5 1 ...

fxch st(1) ; 1 2.5 ...

fld st(1) ; 2.5 1 2.5 ...

; отделим целую часть - очередную слева цифру дробной части исходного числа,-

fprem ; 0.5 1 2.5 ...

; оставим от произведения лишь дробную часть,

fsub st(2), st ; 0.5 1 2 ...

fxch st(2) ; 2 1 0.5 ...

; сохраним полученную цифру во временной ячейке

fistp word ptr [bp - 4] ; 1 0.5 ...

; и сразу выведем.

mov ah, 02h

mov dl, [bp - 4]

add dl, 30h

int 21h

; Теперь, если остаток дробной части ненулевой

fxch st(1) ; 0.5 1 ...

ftst

fstsw ax

sahf

; и мы вывели менее шести цифр, продолжим.

loopnz @of4 ; 0 1 ...

; Итак, число выведено. Осталось убрать мусор из стэка.

@of5: fstp st(0) ; 1 ...

fstp st(0)

; Точнее, стэков.

leave

pop dx

pop cx

pop ax

ret

outfloat endp

exit: ; выход из программы

OutputString pressAnyKeyMessage

mov ah, 08h ; жмем любую кнопку

int 21h

mov ax,4c00h ; завершаем работу программы

int 21h

END start

### *Задание к лабораторной работе 5*

Написать программу эмулятора АЛУ, реализующего *операцию умножения с плавающей точкой* над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

start:

mov ax, @data

mov ds, ax

OutputString inputFirstNumberMessage ;вводим первое число

call InputNumber

fstp firstNumber

OutputString inputSecondNumberMessage ;вводим второе число

call InputNumber

fstp secondNumber

OutputString resultOfProductMessage ; суммируем их

mov eax,firstNumber

mov ebx,secondNumber

call Product

call outfloat

jmp exit

\_oveflow:

OutputString owerflowMessage

jmp exit

\_underflow:

OutputString underflowMessage

jmp exit

Product proc

push ebx

push ecx

push edx

push edi

push esi

;сохраняем операнды

mov edi, eax

mov esi, ebx

push eax

push ebx

;Если значение одного из чисел равно 0, то понятно, какой будет результат

cmp eax,0

je resultIsZero

cmp ebx,0

je resultIsZero

jmp resultIsntZero

resultIsZero:

pop ebx

pop eax

mov eax, 0

jmp endProduct

resultIsntZero:

; получаем мантиссы

; в eax хранится мантисса firstNumber

; в ebx хранится мантисса secondNumber

mov ecx, eax

call GetMantissa

mov eax, ecx

mov ecx, ebx

call GetMantissa

mov ebx, ecx

;получаем порядки чисел

;в edi порядок firstNumber

;в esi порядок secondNumber

mov ecx, edi

call GetExponent

mov edi, ecx

mov ecx, esi

call GetExponent

mov esi, ecx

add edi, esi ; суммируем порядки

;т.к. при хранении порядков используется смещенное представление, то

sub edi,127 ; здесь смещение учли аж 2 раза при сложении. Поэтому вычитаем его

cmp edi, 255

jae \_oveflow

;восстанавливаем изначальные операнды

pop edx

pop ecx

;Проверяем знаки этих операндов

; Узнаем, какой будет знак у произведения

mov esi, 0

test ecx, ecx

jns firstNumberIsPositive

inc si

firstNumberIsPositive:

test edx, edx

jns secondNumberIsPositive

inc si

secondNumberIsPositive:

cmp si,1 ; если SI=0 или SI=2 -- значит знак произведения положительный

jne resultIsPositive

mov esi, 80000000h

resultIsPositive:

mov edx, 0

mul ebx ;Перемножаем манциссы ; результат помещается в edx и eax

mov cx, 15

shiftLeft:

shl eax,1 ; сдвигается мантисса

rcl edx,1 ; циклический сдвиг влево

loop shiftLeft

; в edi порядок находится

inc edi

normalizationOfMantissa:

bt edx, 30

jc orderCheckOwerflow

shl eax, 1

rcl edx, 1

dec edi

jmp normalizationOfMantissa

orderCheckOwerflow:

cmp edi, 255

jae \_oveflow

cmp edi,40

jl \_underflow

;все засовываем в один регистр -- eax

shl edi, 23

shl edx, 2

shr edx, 9

or esi, edi

or esi, edx

mov eax, esi

endProduct:

pop esi

pop edi

pop edx

pop ecx

pop ebx

ret

Product endp

### *Задание к лабораторной работе 6*

Написать программу эмулятора АЛУ, реализующего *операцию деления с плавающей точкой* над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

start:

mov ax, @data

mov ds, ax

OutputString inputFirstNumberMessage ;вводим первое число

call InputNumber

fstp firstNumber

OutputString inputSecondNumberMessage ;вводим второе число

call InputNumber

fstp secondNumber

OutputString resultOfProductMessage ; суммируем их

mov eax,firstNumber

mov ebx,secondNumber

call divide

call outfloat

jmp exit

\_oveflow:

OutputString owerflowMessage

jmp exit

\_underflow:

OutputString underflowMessage

jmp exit

divisionByZero:

OutputString divisionByZeroMessage

jmp exit

Division proc

push ebx

push ecx

push edx

push edi

push esi

;сохраняем операнды

mov edi, eax

mov esi, ebx

push eax

push ebx

cmp eax,0 ; если первое число 0 -- результат 0

je firstNumberIsZero

cmp ebx,0 ; если второе число 0 -- значит ошибка деления на 0

je divisionByZero

jmp continueSum

firstNumberIsZero:

pop ebx

pop eax

mov eax, 0

jmp endDivision

;Суммирование порядков (итог в EDI)

; Извлекаем показатель и проверяем на переполнение(EDI | ESI)

continueSum:

; получаем мантиссы

; в eax хранится мантисса firstNumber

; в ebx хранится мантисса secondNumber

mov ecx, eax

call GetMantissa

mov eax, ecx

mov ecx, ebx

call GetMantissa

mov ebx, ecx

;получаем порядки чисел

;в edi порядок firstNumber

;в esi порядок secondNumber

mov ecx, edi

call GetExponent

mov edi, ecx

mov ecx, esi

call GetExponent

mov esi, ecx

sub edi,esi ; вычитаем порядки

add edi,127 ; получили несмещенный код. надо сместить. Добавляем смещение

cmp edi, 255

jae \_oveflow

cmp edi, 0

jl \_underflow

;восстанавливаем изначальные операнды

pop edx

pop ecx

;Проверяем знаки этих операндов

;и узнаем знак произведения

mov esi, 0

test ecx, ecx

jns firstNumberIsPositive

inc si

firstNumberIsPositive:

test edx, edx

jns secondNumberIsPositive

inc si

secondNumberIsPositive:

cmp si,1 ; если SI=0 или SI=2 -- значит знак произведения положительный

jne resultIsPositive

mov esi, 80000000h

;Делим мантиссы

resultIsPositive:

mov edx, 0

mov cx, 0

littleByteIsZero:

bt ebx,0 ;в ebx мантисса второго числа

jc divis ; нулевой байт равен 1

inc cx ; в cx количество вытолкнутых нулей

shr ebx,1

jmp littleByteIsZero

divis:

mov count,cx

div ebx ; делим мантиссы

mov cx, 23

sub cx, count ; теперь в cx количество невытолкнутых цифр мантиссы

shl eax, 8

cmp cx, 0

je normalizationOfMantissa

shiftLeft:

shl eax,1

loop shiftLeft

; Нормализация манциссы

;inc edi

normalizationOfMantissa:

bt eax, 31

jc owerflowCheck

shl eax, 1

;rcl edx, 1

dec edi

jmp normalizationOfMantissa

owerflowCheck:

cmp edi,40

jl \_underflow

cmp edi, 255

jae \_oveflow

;все засовываем в один регистр -- eax

shl edi, 23

shl eax, 1

shr eax, 9

or esi, edi

or esi, eax

mov eax, esi

endDivision:

pop esi

pop edi

pop edx

pop ecx

pop ebx

ret

Division endp