Учреждение Образования

«Белорусский Государственный Университет Информатики и Радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра Информатики

Дисциплина: АВС

Отчет

По лабораторным работам № 1-3

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ С ЦЕЛЫМИ ЧИСЛАМИ

Выполнил студент

Группы 253505

Волчецкий А.М.

Минск 2014

### *Задание к лабораторной работе 1*

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

.model small

.stack 100h

.data

inputFirstNumberMessage db 10,13,'Print first number(from -128 to 127): $'

inputSecondNumberMessage db 10,13,'Print second number(from -128 to 127): $'

resultOfSumMessage db 10,13,'Sum: $'

resultOfDifferenceMessage db 10,13,'Difference: $'

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

owerflowMessage db 'Owerflow$'

;13 - перевод строки

;10 - переход к началу строки

;$ - конец строки

minus db '- $'

firstNumber dw 0

secondNumber dw 0

maxNumber dw 127

minNumber dw -128

isOwerflowing dw 0 ; если 0 -- переполнения нет, если 1 -- переполнение

.code

.386

; макрос выводит на экран строку

; перед вызовов процедуры строку нужно поместить в стек

OutputString macro string

push ax

mov dx, offset string

mov ah, 9

int 21h

pop ax

endm

WriteSimbol macro simbol

push ax

mov dl, simbol

mov ah, 02h

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

pop ax

endm

; делает проверку на выход числа за допустимый диапазон

; если есть переполнение, то в isOwerflowing храниться 1, если нет -- 0

CheckOfOwerflowing macro number

local checkForNegativeNumber, owerflow, exitFromCheck, numberAlreadyNegative ; хз зачем, но без этого не скомпилиться

push number

mov isOwerflowing, 0 ; переполнения нет

cmp number, 0 ; также число может сюда передастся сразу отрицательным. и в регистре si может быть все, что угодно

jl numberAlreadyNegative

cmp si, 1 ; если равно -- значит число отрицательное

je checkForNegativeNumber

;проверка для положительного числа

cmp number, maxNumber ; проверяем, является ли число больше 127

ja owerflow

jmp exitFromCheck

checkForNegativeNumber: ;проверка для отрицательного числа

neg number

numberAlreadyNegative:

cmp number, minNumber ; проверяем, является ли число меньше -128

jb owerflow

jmp exitFromCheck

owerflow:

mov isOwerflowing, 1 ; переполнение есть

exitFromCheck:

pop number

endm

; если ввели минус -- то число сделается отрицательным

MakeNumberSigned macro number

cmp si, 0

je exitFromMakeNumberSigned

neg number

exitFromMakeNumberSigned:

endm

;модуль числа

AbsNumber macro number

cmp number, 0

jb exitFromAbsNumber

neg number

exitFromAbsNumber:

endm

start: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;тут начало

mov ax, @data

mov ds, ax

OutputString inputFirstNumberMessage

call InputNumber

mov firstNumber, ax

OutputString inputSecondNumberMessage

call InputNumber

mov secondNumber, ax

OutputString resultOfSumMessage

call Sum

call OutputNumber

OutputString resultOfDifferenceMessage

call Difference ; вычитание производится над теми же числами

call OutputNumber

jmp exit

; процедура ввода числа.

; введенное число будет храниться в регистре ax

InputNumber proc

mov ax, 0

mov cx, 0

mov si, 0 ; если здесь храниться 1 -- число отрицательное, 0 -- неотрицательное

mov di, 0 ; здесь храниться количество цифр в числе(минус тоже учитывается)

startInput:

mov ah, 08h ; вводим символ

int 21h

cmp al, minus

je negativeNumber ; переход на метку если был введен минус

cmp al,39h

ja startInput ; если ввели не число, то ввод придется повторить

cmp al,30h ; была нажата системная клавиша(esc, enter, backspace)

jb pressedSystemButton

mov bl, al ; сохраняем символьное представление цифры(чтобы чуть позже вывести его на экран)

mov ax, 10

mul cx ; делаем так, чтобы к промежуточному числу могли добавить еще цифру

mov cx, ax ; результат записываем в cx

mov al, bl

sub al, '0' ; получаем из символа цифру

mov ah, 0h

add cx, ax

CheckOfOwerflowing cx

cmp isOwerflowing, 1

je owerflowing

inc di

WriteSimbol bl

jmp startInput

negativeNumber:

cmp si, 1 ; если минус уже есть то больше его не вводим

je startInput

WriteSimbol minus

mov si, 1

mov di, 1

jmp startInput

owerflowing: ; если переполнение, то ввод придется повторить

mov dx, 0

mov ax, cx ; делимое

mov cx, 10 ; делитель

div cx

mov cx, ax

jmp startInput

pressedSystemButton:

cmp al, 8 ; если была нажата клавиша Backspace

je pressedBackspace

cmp al, 13 ; если была нажата клавиша Enter

je endOfInput

pressedBackspace:

mov dx, 0

mov ax, cx ; делимое

mov cx, 10 ; делитель

div cx

mov cx, ax

cmp di, 0 ; если удалили все число, то дальше удалять нельзя

je reset

dec di

mov dl,8

mov ah,2

int 21h

mov dl,32

int 21h

mov dl,8

int 21h

jmp startInput

reset: ; сюда переходим, когда удалено все число

mov si, 0 ; теперь мы можем опять вводить минус

mov cx, 0

jmp startInput

endOfInput:

MakeNumberSigned cx

mov ax, cx

ret

InputNumber endp

; выводит число, что в регистре ax

OutputNumber proc

push di ; здесь будет храниться количество цифры в числе

; исходное значение регистра помещаем в стек, чтобы в конце процедуры восстановить его

cmp isOwerflowing, 1 ; если была обнаружено переполнение, то число не выводим

je exitFromOutputNumber

mov di, 0

cmp ax, 0

jge digitToStack ; если число неторицательное, то переходим на эту метку

WriteSimbol minus ; выводим на экран минус

AbsNumber ax ; далее работаем с модулем числа

digitToStack: ; получаем цифры числа и загоняем их в стек

mov dx, 0

mov cx, 10

div cx

push dx ; остаток от деление в стек

inc di

cmp ax, 0 ;если цифры в числе остались, то

jne digitToStack ; продолжаем их получать

digitFromStack: ; теперь достаем цифры из стека и выводим их

pop dx

add dx, '0' ; делаем из числа символ

WriteSimbol dl

dec di

cmp di, 0

jne digitFromStack

exitFromOutputNumber:

pop di

ret

OutputNumber endp

;суммирует 2 числа, которые находятся в переменных firstNumber и secondNumber

; результат находится в ax

Sum proc

push cx

push firstNumber

push secondNumber

mov si, 0 ; косяк. надо обнулять переменную

startSum:

mov ax, firstNumber

mov cx, firstNumber ; ниже будет xor и содержимое ax изменится.

;поэтому копируем еще и сюда значение переменной

mov bx, secondNumber

xor ax, bx ; сложение по модулю 2

and cx, bx ; логическое И

shl cx, 1 ; выполняем сдвиг разрядов на единицу

mov firstNumber, ax ; в а будет храниться результат xor

mov secondNumber, cx ; в secondNumber будет храниться результат сдвига разрядов в числе

; полученном в результате операции and

cmp secondNumber, 0

je owerflowCheck

cmp firstNumber, 0

jne startSum

mov ax, secondNumber ; т.к. в данном случае firstNumber = 0

owerflowCheck:

CheckOfOwerflowing ax

cmp isOwerflowing, 0

je exitFromSum

OutputString owerflowMessage

exitFromSum:

pop secondNumber

pop firstNumber

pop cx

ret

Sum endp

; отнимает от числа в переменной firstNumber число в переменной secondNumber

Difference proc

push firstNumber

push secondNumber

mov cx, firstNumber

mov bx, secondNumber

; для перехода от разности к сложению производим отрицание числа

not bx

cmp bx, 127

jne b

mov ax, 128

jmp a

b:

mov firstNumber, 1

mov secondNumber, bx

call Sum

a:

; теперь числа можно просто складывать

mov firstNumber, cx

mov secondNumber, ax

call Sum

pop secondNumber

pop firstNumber

ret

Difference 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 2*

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

start: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;тут начало

mov ax, @data

mov ds, ax

OutputString inputFirstNumberMessage

call InputNumber

mov firstNumber, ax

OutputString inputSecondNumberMessage

call InputNumber

mov secondNumber, ax

OutputString resultOfProductMessage

call Product

call OutputNumber

CheckOfOwerflowing8 ax

cmp isOwerflowing, 1

je owerflow1

jmp exit

owerflow1:

OutputString owerflowMessage

jmp exit

; произведение чисел

; первое число храниться в firstNumber, второе -- в secondNumber

; результат в ax

Product proc

mov ax, firstNumber

mov bx, secondNumber

WhatIsSignOfProduct ax, bx ; определяет знак произведения и записыва его в si

;если в si будет 0 -- тогда результат положителен

push si

mov si, 0

AbsNumber ax ; делаем из этих чисел числа без знака

AbsNumber bx

pop si

mov cx, 0 ; тут будет храниться промежуточный результат

startProduct:

shr bx, 1 ; крайний правый бит будет во флаге CF

jae byteIsZero

push ax

mov firstNumber, cx

mov secondNumber, ax

call Sum

mov cx, ax

pop ax

byteIsZero: ; ничего не прибавляем(т.к. умножаем на 0)

shl ax, 1

cmp bx, 0 ; когда второе число станет равно 0, тогда прекращаем умножение

jne startProduct

; здесь учитывается знак результата произведения

cmp si, 0

je numberIsPositive

neg cx

numberIsPositive:

mov ax, cx

jmp exitFromProduct

owerflowInProduct:

OutputString owerflowMessage

jmp exit

exitFromProduct:

CheckOfOwerflowing8 ax

cmp isOwerflowing, 1

je owerflowInProduct

ret

Product endp

### *Задание к лабораторной работе 3*

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

start: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;тут начало

mov ax, @data

mov ds, ax

OutputString inputFirstNumberMessage

call InputNumber

mov firstNumber, ax

OutputString inputSecondNumberMessage

call InputNumber

mov secondNumber, ax

OutputString resultOfProductMessage

cmp secondNumber, 0

jne allComplete

OutputString divisionByZeroMessage

jmp outputRemainder

allComplete:

call Division

call OutputNumber

outputRemainder:

OutputString remainderMessage

mov ax, cx

call OutputNumber

jmp exit

Division proc

mov cx, 0

mov dx, 0

mov bp, 8 ;столько раз будут выполнятся пункты 2-4(сколько разрядов в числе столько раз и повторяется)

point1:

; загружаем делитель

mov bx, secondNumber

; загружаем правую часть делимого

mov dx, firstNumber

mov cl, dl

; загружаем левую часть делимого

; если число положительное, то заполняем нулями ch, если отрицательное--единицами

cmp cl, 0

jge numberIsPositive

mov ch, 11111111b

jmp checkSign

numberIsPositive:

mov ch, 0

checkSign:

WhatIsSignOfDivision bx, cx ;определяем знак результата деления. это понадобиться на 6 пункте

point2:

shl cx, 1

point3:

mov dh, ch ; начальное значение в ch понадобиться в 4 пункте

;в firstNumber и secondNumber будут храниться значения ch и bl соответственно

;чуть ниже эти переменные будут использоваться для сложения или разности

mov ax, 0

mov al, ch

mov firstNumber, ax

mov ax, 0

mov al, bl

mov secondNumber, ax

; сравниваем знаки чисел в ch и bl

mov ah, ch

xor ah, bl

jns signsAreSame;знаки одинаковые. после xor в старшем бите ah будет 0

signsAreDifference:

call Sum

jmp endOfThirdPoint

signsAreSame:

call Difference

endOfThirdPoint:

mov ch, al ; результат суммы/разности сюда сохраняем

point4:

xor al, dh ; проверяем поменялся ли знак ch. в al -- новое значение ch, в dh -- старое

jns сxSignDidntChange ; если операция была успешной, то сразу переходим на эту метку

сxSignChanged: ; тут идет проверка на то, что и в ah и в al находится 0

mov ax, cx

or ah, al ; если сложить 0 или 0 то получиться 0. это позволит выполнить всего одну проверку

cmp ah, 0

je сxSignDidntChange ; когда в ah и al 0 мы тоже переходим на эту метку

;раз дошли сюда -- то предыдущая операция была неуспешной и содержимое регистров сh и сl не равно нулю

; устанавливаем код 0 в cl

shr cl, 1 ;

jnc noperenos

shl cl, 1

or cl, 1b

jmp reclaim

noperenos: ; если ноль уже был в младшем разряде, то просто смещаем назад

shl cl, 1

reclaim:

mov ch, dh ; восстанавливаем прежнее значение ch

jmp point5

сxSignDidntChange: ; устанавливаем код 1 в младшем разряде al

or cl, 1b

point5:

dec bp

cmp bp, 0

jne point2

point6:

push cx

mov ch, 0

cmp si, 0 ; тут знак деления. его определяем на 1 пункте

je resultInAx

neg cl

mov ch, 11111111b

resultInAx:

mov ax, cx

CheckOfOwerflowing ax

cmp isOwerflowing, 0

je exitFromDivision

OutputString owerflowMessage

exitFromDivision:

pop dx

mov cx, 0

mov cl, dh

Convert8to16 cx

push cx

push bx

AbsNumber cx

AbsNumber bx

cmp cx, bx

pop bx

pop cx

jne exitexit

mov firstNumber, ax

cmp bx, 0

jg bxPositive

mov secondNumber, 1

jmp next

bxPositive:

mov secondNumber, -1

next:

call Sum

mov cx, 0

exitexit:

ret

Division endp