МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

# Кафедра проектирования информационно-компьютерных систем

Отчет по предмету

Конструирование программ

Лабораторная работа №1

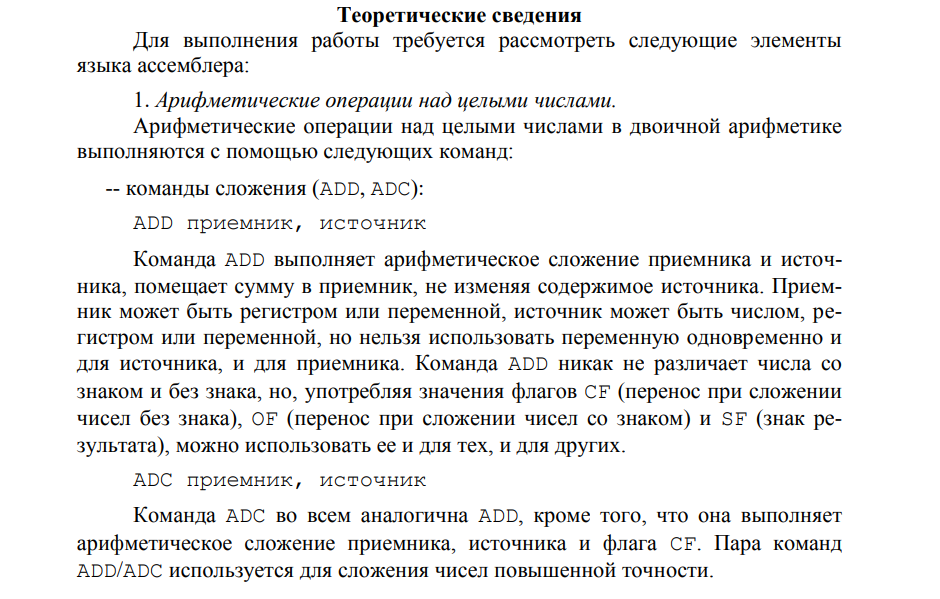
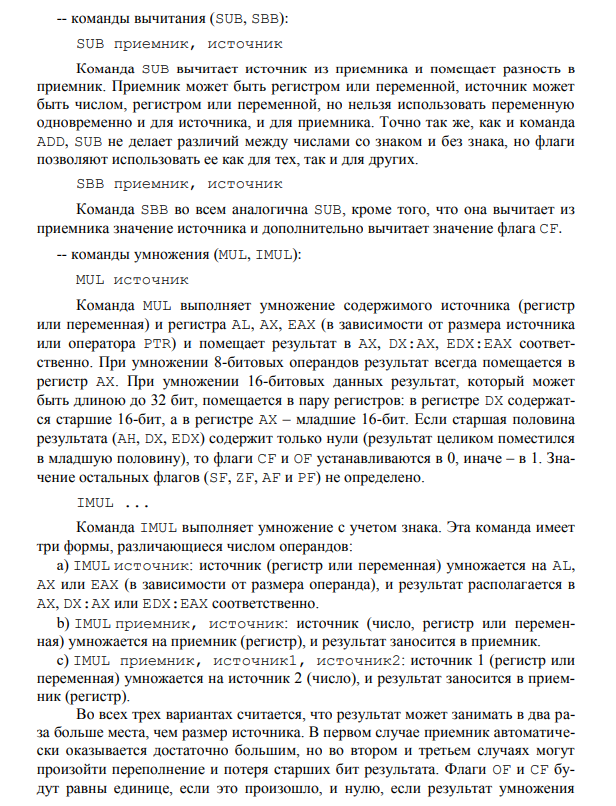
**«**Целочисленные арифметические операции. Обработка массивов числовых данных**»**

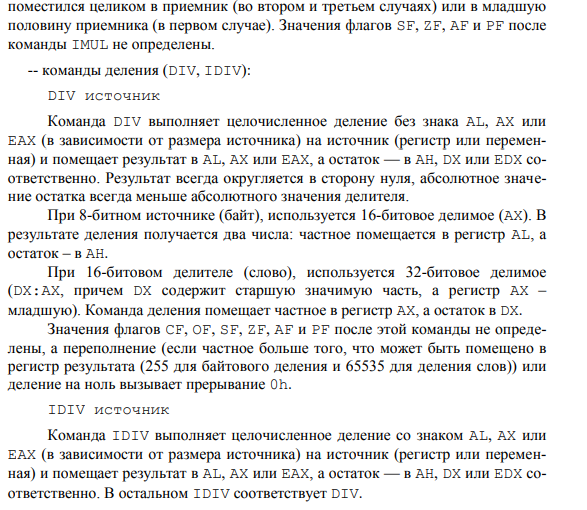
|  |  |
| --- | --- |
| **Выполнил:**  Студент группы 153501  Миролюбов И.И. | **Проверил:**  Туровец Н.О. |

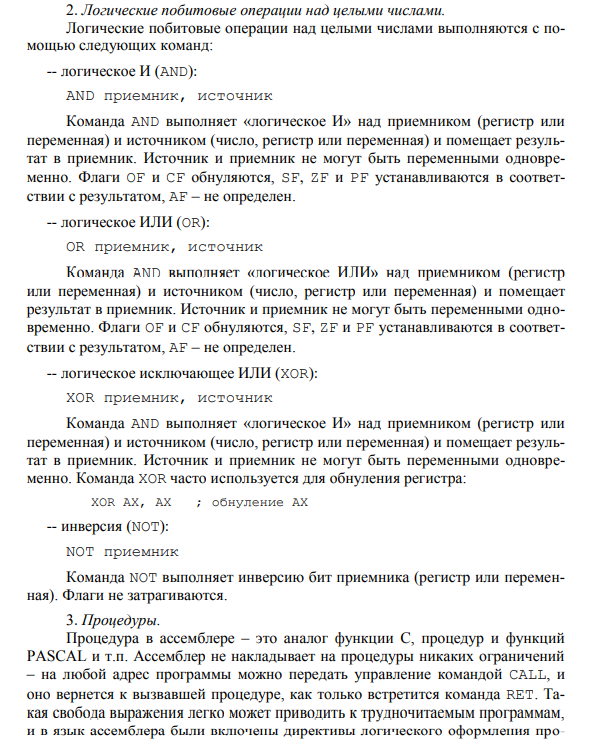
Минск 2022

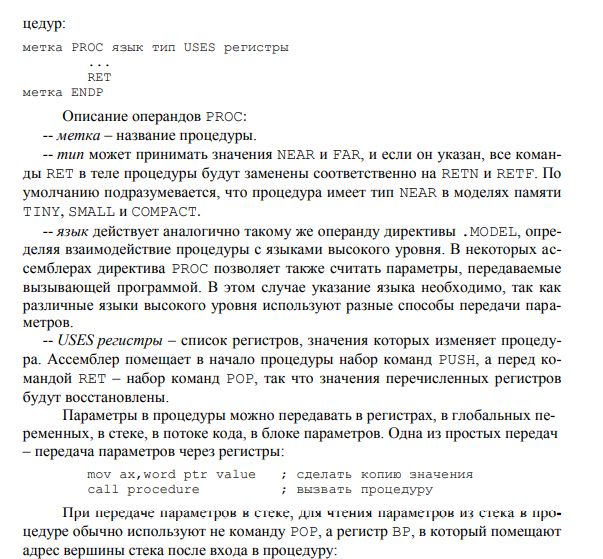
**Цель работы**:

Ознакомиться с арифметическими операциями над целочисленными данными, обработкой массивов чисел, ознакомиться с правилами оформления ассемблерных процедур

**  
**

****

****



**Задание**

15.Ввести массив целых чисел размерностью 30 элементов. Выполнить операцию над всеми элементами массива (оператор может ввести одну из доступных операций: инверсия, модуль, возведение в квадрат, 1/число)

**Код программы**

**.model small**

**.stack 100h**

**.data**

**; operations**

**op db 2,2 dup(?)**

**op1 db 10,13,"1) Print array$"**

**op2 db 10,13,"2) Inverse$"**

**op3 db 10,13,"3) Absolute$:"**

**op4 db 10,13,"4) Square$"**

**op5 db 10,13,"5) Multiplicative inverse$"**

**op6 db 10,13,"6) Quit$"**

**;messages**

**msg1 db 10,13,"Error: incorrect input $"**

**msg2 db 10,13,"Error: overflow $"**

**msg3 db 10,13,"Error: 0 dose not have multiplicative inverse$"**

**msg4 db "Enter array of 30 numbers$"**

**msg5 db 10,13,"Enter number of operation$"**

**;other data**

**nl db 10,13,'$'**

**buf db 10,10 dup(?) ; -32768 to 32767**

**out\_buf db 6 dup(?),'$'**

**mul\_inv\_buf db "00.", 5 dup('0'), '$'**

**array dw 30 dup(?)**

**.code**

**print\_str macro out\_str**

**mov dx,offset out\_str**

**mov ah,9h**

**int 21h**

**endm**

**get\_str macro buff**

**mov ah,0Ah**

**mov dx,offset buff**

**int 21h**

**endm**

**start:**

**mov ax,@data**

**mov ds,ax**

**print\_str msg4**

**print\_str nl**

**call get\_array**

**main:**

**print\_str nl**

**print\_str op1**

**print\_str op2**

**print\_str op3**

**print\_str op4**

**print\_str op5**

**print\_str op6**

**print\_str msg5**

**print\_str nl**

**get\_str op**

**mov bl,op[2]**

**cmp bl,'1'**

**je call\_op1**

**cmp bl,'2'**

**je call\_op2**

**cmp bl,'3'**

**je call\_op3**

**cmp bl,'4'**

**je call\_op4**

**cmp bl,'5'**

**je call\_op5**

**cmp bl,'6'**

**je exit\_main**

**jmp main**

**call\_op1:**

**call print\_array**

**jmp main**

**call\_op2:**

**call inv\_all**

**jmp main**

**call\_op3:**

**call abs\_all**

**jmp main**

**call\_op4:**

**call sqr\_all**

**jmp main**

**call\_op5:**

**call mul\_inv\_all**

**jmp main**

**exit\_main:**

**mov ah,4ch**

**int 21h**

**;;---------------------------------------------------------------**

**inv proc uses ax**

**mov ax,array[di]**

**cmp ax,32768**

**je inv\_error**

**not ax**

**add ax,1**

**mov array[di],ax**

**jmp inv\_end**

**inv\_error:**

**print\_str msg2**

**inv\_end:**

**ret**

**inv endp**

**inv\_all proc uses di**

**mov di,0**

**ia:**

**cmp di,60**

**je ia\_end**

**call inv**

**add di,2**

**jmp ia**

**ia\_end:**

**ret**

**inv\_all endp**

**mul\_inv proc uses ax,bx,cx,dx,si ; multiplicative inverse with 4-5 digits accuracy**

**mov ax,array[di]**

**cmp ax,0**

**je mi\_zero**

**xor bx,bx**

**mov bx, offset mul\_inv\_buf**

**mov si,0**

**fill\_zero:**

**cmp si,8**

**je mi\_check\_sign**

**cmp si,2**

**je dot**

**mov bx[si],'0'**

**inc si**

**jmp fill\_zero**

**dot:**

**inc si**

**jmp fill\_zero**

**mi\_check\_sign:**

**mov bx, offset mul\_inv\_buf + 8**

**mov byte ptr [bx],'$'**

**and ax,32768**

**cmp ax,32768**

**je mi\_negative**

**jmp mi\_positive**

**mi\_negative:**

**mov ax,array[di]**

**not ax**

**add ax,1**

**mov bl,'-'**

**mov mul\_inv\_buf,bl**

**cmp ax,1**

**je mi\_one**

**cmp ax,10000**

**ja more\_10000**

**jmp mi\_main**

**mi\_positive:**

**mov ax,array[di]**

**cmp ax,1**

**je mi\_one**

**cmp ax,10000**

**ja more\_10000**

**mi\_main:**

**xor dx,dx**

**mov bx,ax**

**mov ax,10000**

**idiv bx**

**mov cx,10**

**xor bx,bx**

**mov bx,offset mul\_inv\_buf**

**mov si,6**

**mi\_cycle:**

**xor dx,dx**

**div cx**

**cmp si,2**

**je mi\_main\_end**

**add dl,'0'**

**mov bx[si],dl**

**dec si**

**jmp mi\_cycle**

**mi\_main\_end:**

**print\_str mul\_inv\_buf**

**jmp mi\_end**

**more\_10000:**

**xor dx,dx**

**mov bx,10**

**div bx**

**mov bx,ax**

**mov ax,10000**

**xor dx,dx**

**div bx**

**add al,'0'**

**mov mul\_inv\_buf[7],al**

**print\_str mul\_inv\_buf**

**jmp mi\_end**

**mi\_zero:**

**print\_str msg3**

**jmp mi\_end**

**mi\_one:**

**mov bl,'1'**

**mov mul\_inv\_buf[1],bl**

**print\_str mul\_inv\_buf**

**mi\_end:**

**ret**

**mul\_inv endp**

**mul\_inv\_all proc uses di**

**mov di,0**

**mia:**

**cmp di,60**

**je mia\_end**

**print\_str nl**

**call mul\_inv**

**add di,2**

**jmp mia**

**mia\_end:**

**ret**

**mul\_inv\_all endp**

**sqr proc uses ax**

**mov ax,array[di]**

**and ax,32768**

**cmp ax,32768**

**mov ax,array[di]**

**jne sqr\_main**

**not ax**

**add ax,1**

**sqr\_main:**

**imul ax**

**jo sqr\_error**

**mov array[di],ax**

**jmp sqr\_end**

**sqr\_error:**

**print\_str msg2**

**sqr\_end:**

**ret**

**sqr endp**

**sqr\_all proc uses di**

**mov di,0**

**sa:**

**cmp di,60**

**je sa\_end**

**call sqr**

**add di,2**

**jmp sa**

**sa\_end:**

**ret**

**sqr\_all endp**

**abs proc uses ax**

**mov ax,array[di]**

**cmp ax,32768**

**je abs\_error**

**and ax,32768**

**cmp ax,32768**

**jne abs\_end ;if positive**

**mov ax,array[di]**

**not ax**

**add ax,1**

**mov array[di],ax**

**jmp abs\_end**

**abs\_error:**

**print\_str msg2**

**abs\_end:**

**ret**

**abs endp**

**abs\_all proc uses di**

**mov di,0**

**aa:**

**cmp di,60**

**je aa\_end**

**call abs**

**add di,2**

**jmp aa**

**aa\_end:**

**ret**

**abs\_all endp**

**print\_num proc uses ax,bx,cx,dx,si**

**xor si,si**

**mov si,5**

**mov cx, 10**

**check\_sign:**

**mov ax, array[di]**

**and ax,32768**

**cmp ax,32768**

**je negative**

**jmp positive**

**negative:**

**mov ax,array[di]**

**not ax**

**add ax,1**

**mov bl,'-'**

**mov out\_buf,bl**

**jmp cycle1**

**positive:**

**mov ax,array[di]**

**mov bl,'0'**

**mov out\_buf,bl**

**jmp cycle1**

**cycle1:**

**xor bx,bx**

**mov bx, offset out\_buf**

**xor dx,dx**

**;mov dl, 10**

**div cx**

**add dx,'0'**

**mov bx[si], dl**

**dec si**

**cmp si,0**

**je end\_print\_num**

**jmp cycle1**

**end\_print\_num:**

**print\_str nl**

**print\_str out\_buf**

**ret**

**print\_num endp**

**print\_array proc uses di**

**mov di,0**

**print:**

**cmp di,60**

**je print\_end**

**call print\_num**

**add di,2**

**jmp print**

**print\_end:**

**ret**

**print\_array endp**

**get\_num proc uses ax,bx,cx,dx,si**

**xor cx,cx**

**mov cl, buf[1]**

**cmp cx,0**

**je error\_input ;no input (pressed enter)**

**mov si,2**

**xor bx,bx**

**mov bl,buf[si]**

**cmp bl,'-'**

**jne atoi**

**cmp cl,1**

**je error\_input**

**inc si**

**dec cx**

**atoi:**

**xor ax, ax ; zero a "result so far"**

**top:**

**cmp cx,0**

**je end\_atoi**

**xor bx,bx**

**mov dx, 10**

**mov bl, buf[si] ; get a character**

**inc si ; ready for next one**

**cmp bl, '0' ; valid?**

**jb error\_input**

**cmp bl, '9'**

**ja error\_input**

**sub bl, '0' ; "convert" character to number**

**imul dx ; multiply "result so far" by ten**

**jc error\_overflow**

**add ax, bx ; add in current digit**

**dec cx**

**jmp top ; until done**

**end\_atoi:**

**xor bx,bx**

**mov bl,buf[2]**

**cmp bl,'-'**

**je \_negative**

**cmp ax,32767**

**ja error\_overflow**

**jmp put\_in\_array**

**\_negative:**

**cmp ax, 32768**

**ja error\_overflow**

**not ax ; turn into two's complement**

**add ax,1 ;**

**put\_in\_array:**

**mov array[di],ax**

**jmp exit**

**error\_input:**

**sub di,2**

**print\_str msg1**

**jmp exit**

**error\_overflow:**

**sub di,2**

**print\_str msg2**

**;jmp exit**

**exit:**

**ret**

**get\_num endp**

**get\_array proc uses di**

**xor di,di**

**get\_array\_loop:**

**get\_str buf**

**call get\_num**

**add di,2**

**print\_str nl**

**cmp di, 60**

**jne get\_array\_loop**

**ret**

**get\_array endp**

**end start**

**Вывод**

В ходе выполнения лабораторной работы были изучены арифметические операции над целочисленными данными, обработка массивов чисел, правила оформления ассемблерных процедур. Так же в данной лабораторной работе была разработана программа, которая выполняет операцию над всеми элементами массива (оператор может ввести одну из доступных операций: инверсия, модуль, возведение в квадрат, 1/число)