МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

# Кафедра проектирования информационно-компьютерных систем

Отчет по предмету

Программирование на языке ассемблера

Лабораторная работа №3

**«ЦЕЛОЧИСЛЕННЫЕ АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ.  
ОБРАБОТКА МАССИВОВ ЧИСЛОВЫХ ДАННЫХ»**

|  |  |
| --- | --- |
| **Выполнил:**  Студент группы 150501  Смоленский Н.О. | **Проверил:**  Туровец Н.О. |

Минск 2022

**Цель работы**: ознакомиться с арифметическими операциями над целочисленными данными, обработкой массивов чисел, ознакомиться с правилами оформления ассемблерных процедур.

**Теоретические сведения:**

1. *Арифметические операции над целыми числами.*

Арифметические операции над целыми числами в двоичной арифметике выполняются с помощью следующих команд:

-- команды сложения (ADD, ADC):

ADD приемник, источник

Команда ADD выполняет арифметическое сложение приемника и источника, помещает сумму в приемник, не изменяя содержимое источника. Приемник может быть регистром или переменной, источник может быть числом, регистром или переменной, но нельзя использовать переменную одновременно и для источника, и для приемника. Команда ADD никак не различает числа со знаком и без знака, но, употребляя значения флагов CF (перенос при сложении чисел без знака), OF (перенос при сложении чисел со знаком) и SF (знак результата), можно использовать ее и для тех, и для других.

ADC приемник, источник

Команда ADC во всем аналогична ADD, кроме того, что она выполняет  
арифметическое сложение приемника, источника и флага СF. Пара команд  
ADD/ADC используется для сложения чисел повышенной точности.

-- команды вычитания (SUB, SBB):

SUB приемник, источник

Команда SUB вычитает источник из приемника и помещает разность в  
приемник. Приемник может быть регистром или переменной, источник может быть числом, регистром или переменной, но нельзя использовать переменную одновременно и для источника, и для приемника. Точно так же, как и команда ADD, SUB не делает различий между числами со знаком и без знака, но флаги позволяют использовать ее как для тех, так и для других.

SBB приемник, источник

Команда SBB во всем аналогична SUB, кроме того, что она вычитает из приемника значение источника и дополнительно вычитает значение флага CF.

-- команды умножения (MUL, IMUL):

MUL источник

Команда MUL выполняет умножение содержимого источника (регистр  
или переменная) и регистра AL, АХ, ЕАХ (в зависимости от размера источника или оператора PTR) и помещает результат в АХ, DX:AX, EDX:EAX соответственно.

IMUL ...

Команда IMUL выполняет умножение с учетом знака. Эта команда имеет три формы, различающиеся числом операндов:

a) IMUL источник: источник (регистр или переменная) умножается на AL, АХ или ЕАХ (в зависимости от размера операнда), и результат располагается в АХ, DX:AX или EDX:EAX соответственно.

b) IMUL приемник, источник: источник (число, регистр или переменная) умножается на приемник (регистр), и результат заносится в приемник.

c) IMUL приемник, источник1, источник2: источник 1 (регистр или переменная) умножается на источник 2 (число), и результат заносится в приемник (регистр).

-- команды деления (DIV, IDIV):

DIV источник

Команда DIV выполняет целочисленное деление без знака AL, АХ или  
ЕАХ (в зависимости от размера источника) на источник (регистр или переменная) и помещает результат в AL, АХ или ЕАХ, а остаток — в АН, DX или EDX соответственно. Результат всегда округляется в сторону нуля, абсолютное значение остатка всегда меньше абсолютного значения делителя.

При 8-битном источнике (байт), используется 16-битовое делимое (AX). В результате деления получается два числа: частное помещается в регистр AL, а остаток – в AH.

При 16-битовом делителе (слово), используется 32-битовое делимое  
(DX:AX, причем DX содержит старшую значимую часть, а регистр AX –  
младшую). Команда деления помещает частное в регистр AX, а остаток в DX.

IDIV источник

Команда IDIV выполняет целочисленное деление со знаком AL, АХ или ЕАХ (в зависимости от размера источника) на источник (регистр или переменная) и помещает результат в AL, АХ или ЕАХ, а остаток — в АН, DX или EDX соответственно. В остальном IDIV соответствует DIV.

2. *Логические побитовые операции над целыми числами.*

Логические побитовые операции над целыми числами выполняются с помощью следующих команд:

-- логическое И (AND):

AND приемник, источник

Команда AND выполняет «логическое И» над приемником (регистр или переменная) и источником (число, регистр или переменная) и помещает результат в приемник. Источник и приемник не могут быть переменными одновременно. Флаги OF и CF обнуляются, SF, ZF и PF устанавливаются в соответствии с результатом, AF – не определен.

-- логическое ИЛИ (OR):

OR приемник, источник

Команда AND выполняет «логическое ИЛИ» над приемником (регистр  
или переменная) и источником (число, регистр или переменная) и помещает  
результат в приемник. Источник и приемник не могут быть переменными одновременно. Флаги OF и CF обнуляются, SF, ZF и PF устанавливаются в соответствии с результатом, AF – не определен.

-- логическое исключающее ИЛИ (XOR):

XOR приемник, источник

Команда AND выполняет «логическое И» над приемником (регистр или переменная) и источником (число, регистр или переменная) и помещает результат в приемник. Источник и приемник не могут быть переменными одновременно. Команда XOR часто используется для обнуления регистра: XOR AX, AX ; обнуление AX

-- инверсия (NOT):

NOT приемник

Команда NOT выполняет инверсию бит приемника (регистр или перемен-  
ная). Флаги не затрагиваются.

3. *Процедуры.*

Процедура в ассемблере – это аналог функции C, процедур и функций PASCAL и т.п. Ассемблер не накладывает на процедуры никаких ограничений – на любой адрес программы можно передать управление командой CALL, и оно вернется к вызвавшей процедуре, как только встретится команда RET. Такая свобода выражения легко может приводить к трудночитаемым программам, и в язык ассемблера были включены директивы логического оформления процедур:

метка PROC язык тип USES регистры

...

RET

метка ENDP

Описание операндов PROC:

-- метка – название процедуры.

-- тип может принимать значения NEAR и FAR, и если он указан, все команды RET в теле процедуры будут заменены соответственно на RETN и RETF. По умолчанию подразумевается, что процедура имеет тип NEAR в моделях памяти TINY, SMALL и COMPACT.

-- язык действует аналогично такому же операнду директивы .MODEL, определяя взаимодействие процедуры с языками высокого уровня. В некоторых ассемблерах директива PROC позволяет также считать параметры, передаваемые вызывающей программой. В этом случае указание языка необходимо, так как различные языки высокого уровня используют разные способы передачи параметров.

-- USES регистры – список регистров, значения которых изменяет процедура. Ассемблер помещает в начало процедуры набор команд PUSH, а перед командой RET – набор команд POP, так что значения перечисленных регистров будут восстановлены.

**Код программы:**

*.model small*

*.stack 100h*

*.data*

*size equ 10*

*size\_array equ 5*

*array dw size\_array dup(?)*

*current\_element dw ?*

*sign dw ?*

*string db size dup('$')*

*ten dw 10*

*minus dw -1*

*min dw ?*

*max dw ?*

*difference dw ?*

*result dw ?*

*msg1 db "Enter the array: $"*

*msg2 db "Entered array: $"*

*msg3 db "Difference of maximum and minimum value: $"*

*msg4 db "Result: $"*

*msg5 db "Min: $"*

*msg6 db "Max: $"*

*msg7 db "Overflow!",0Dh,0Ah,'$'*

*msg8 db "Error: Divided by zero!",0Dh,0Ah,'$'*

*error db "Incorrect input!",0Dh,0Ah,'$'*

*newl db 0Dh,0Ah,'$'*

*min\_num equ 32768*

*max\_num equ 32767*

*.code*

*Input\_string macro str*

*lea dx,str*

*mov offset str,size-3*

*mov ah,0Ah*

*int 21h*

*mov ax,0*

*call Check\_num*

*endm*

*Output\_string proc*

*mov ah,9*

*int 21h*

*ret*

*Output\_string endp*

*Overflow\_output proc*

*mov dx,offset newl*

*call Output\_string*

*mov dx,offset msg7*

*call Output\_string*

*mov dx,offset newl*

*call Output\_string*

*call Exit*

*ret*

*Overflow\_output endp*

*Output proc*

*Outp:*

*inc di*

*mov dx,0*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*div ten*

*jo call Overflow\_output*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*add dx,'0'*

*push dx*

*cmp ax,0*

*jne Outp*

*cmp sign,-1*

*jne Output\_num*

*Output\_sumb '-'*

*Output\_num:*

*pop ax*

*Output\_sumb ax*

*dec di*

*cmp di,0*

*jne loop Output\_num*

*ret*

*Output endp*

*Output\_sumb macro symb*

*mov dx,symb*

*mov ah,06h*

*int 21h*

*endm*

*Negative\_sign proc*

*mov sign,-1*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*mul minus*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*mov di,0*

*call Output*

*ret*

*Negative\_sign endp*

*Positive\_sign proc*

*mov sign,1*

*mov di,0*

*call Output*

*ret*

*Positive\_sign endp*

*Check\_num proc*

*mov si,offset string*

*add si,2*

*cmp [si],'-'*

*je Negative\_num*

*mov bx,0*

*mov sign,1*

*call Convert\_num*

*ret*

*Negative\_num:*

*mov sign,-1*

*inc si*

*mov bx,0*

*call Convert\_num*

*ret*

*Check\_num endp*

*Convert\_num proc*

*cmp [si],'0'*

*jb call Num\_error*

*cmp [si],'9'*

*ja call Num\_error*

*mov ax,ten*

*mul bx*

*mov bx,ax*

*push bx*

*mov ax,0*

*mov al,[si]*

*sub ax,'0'*

*pop bx*

*add bx,ax*

*inc si*

*cmp [si],0Dh*

*jne call Convert\_num*

*mov ax,bx*

*cmp sign,-1*

*je min\_point*

*jne max\_point*

*max\_point:*

*cmp ax,max\_num*

*ja call Num\_error*

*jmp point*

*min\_point:*

*cmp ax,min\_num*

*ja call Num\_error*

*point:*

*mul sign*

*mov current\_element,ax*

*ret*

*Convert\_num endp*

*Num\_error proc*

*lea dx,newl*

*call Output\_string*

*lea dx,error*

*call Output\_string*

*Input\_string string*

*ret*

*Num\_error endp*

*Exit proc*

*mov ax,4C00h*

*int 21h*

*ret*

*Exit endp*

*start:*

*mov ax,@data*

*mov ds,ax*

*mov cx,size\_array*

*mov di,offset array*

*push es*

*push di*

*mov dx,offset msg1*

*call Output\_string*

*mov dx,offset newl*

*call Output\_string*

*Input\_array:*

*Input\_string string*

*mov dx,offset newl*

*call Output\_string*

*mov ax,current\_element*

*stosw*

*loop Input\_array*

*mov cx,size\_array*

*mov di,0*

*mov dx,offset msg2*

*call Output\_string*

*pop si*

*mov bx,ds*

*mov es,bx*

*pop ds*

*cld*

*Output\_array:*

*mov ax,0*

*lodsw*

*cmp ax,0*

*js negative*

*call Positive\_sign*

*jmp continue*

*negative:*

*call Negative\_sign*

*continue:*

*Output\_sumb ' '*

*loop Output\_array*

*mov di,size\_array*

*mov si,offset array*

*mov ax,0*

*lodsw*

*dec di*

*mov min,ax*

*jmp Find\_min*

*Change\_min:*

*mov min,ax*

*Find\_min:*

*cmp di,0*

*jbe Find\_continue*

*mov ax,0*

*lodsw*

*dec di*

*cmp min,ax*

*jg Change\_min*

*cmp di,0*

*jge Find\_min*

*Find\_continue:*

*mov di,size\_array*

*mov si,offset array*

*mov ax,0*

*lodsw*

*dec di*

*mov max,ax*

*jmp Find\_max*

*Change\_max:*

*mov max,ax*

*Find\_max:*

*cmp di,0*

*jbe Find\_end*

*mov ax,0*

*lodsw*

*dec di*

*cmp max,ax*

*jl Change\_max*

*cmp di,0*

*jge Find\_max*

*Find\_end:*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*mov dx,offset newl*

*call Output\_string*

*mov dx,offset msg5*

*call Output\_string*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*mov ax,min*

*mov cx,1*

*cmp ax,0*

*js negative2*

*call Positive\_sign*

*jmp continue2*

*negative2:*

*call Negative\_sign*

*continue2:*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*mov dx,offset newl*

*call Output\_string*

*mov dx,offset msg6*

*call Output\_string*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*mov ax,max*

*mov cx,1*

*cmp ax,0*

*js negative3*

*call Positive\_sign*

*jmp continue3*

*negative3:*

*call Negative\_sign*

*continue3:*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*mov dx,offset newl*

*call Output\_string*

*mov dx,offset msg3*

*call Output\_string*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*mov ax,max*

*sub ax,min*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*jo call Overflow\_output*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*mov difference,ax*

*mov ax,difference*

*mov cx,1*

*cmp ax,0*

*js negative4*

*call Positive\_sign*

*jmp continue4*

*negative4:*

*call Negative\_sign*

*continue4:*

*mov sign,1*

*cmp max,0*

*js change\_sign*

*jmp cont*

*change\_sign:*

*mov ax,max*

*mov sign,-1*

*mul sign*

*mov max,ax*

*cont:*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*mov dx,offset newl*

*call Output\_string*

*mov dx,offset msg4*

*call Output\_string*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*cmp max,0*

*je divby0*

*cmp sign,-1*

*jne cont2:*

*Output\_sumb '-'*

*cont2:*

*mov ax,difference*

*mov dx,0*

*div max*

*jo call Overflow\_output*

*mov result,ax*

*mov cx,1*

*cmp ax,0*

*js negative5*

*call Positive\_sign*

*jmp continue5*

*negative5:*

*call Negative\_sign*

*continue5:*

*Output\_sumb '.'*

*mov ax,max*

*mul result*

*mov dx,ax*

*mov ax,difference*

*sub ax,dx*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*mul ten*

*jo call Overflow\_output*

*mul ten*

*jo call Overflow\_output*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*mov dx,0*

*div max*

*jo call Overflow\_output*

*mov cx,1*

*cmp ax,0*

*call Positive\_sign*

*call Exit*

*divby0:*

*push ds*

*mov bx,es*

*pop es*

*mov ds,bx*

*mov dx,offset newl*

*call Output\_string*

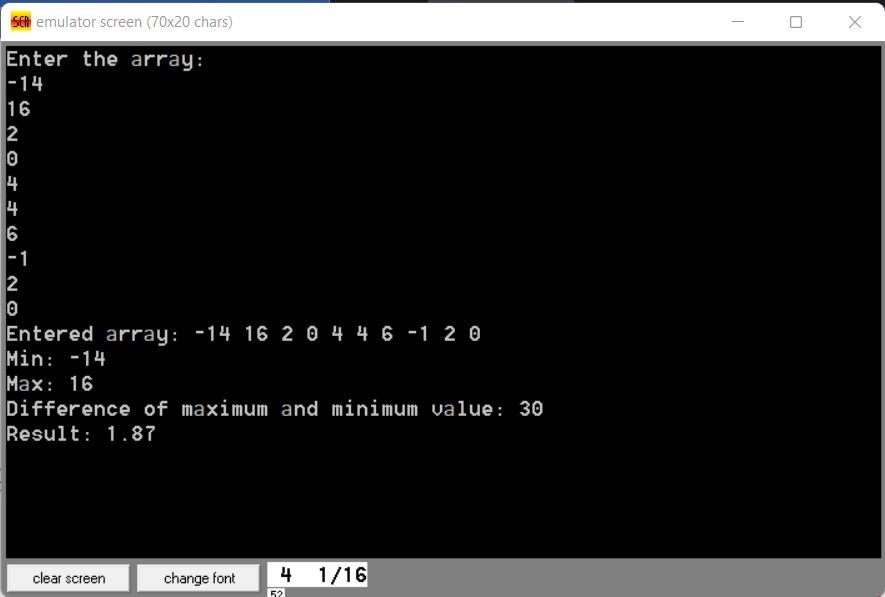
*mov dx,offset msg8*

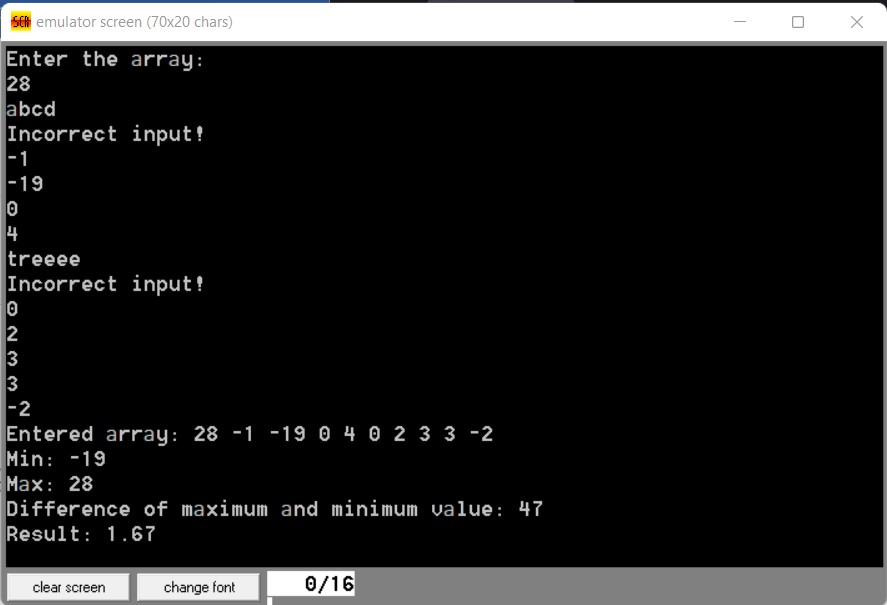
*call Output\_string*

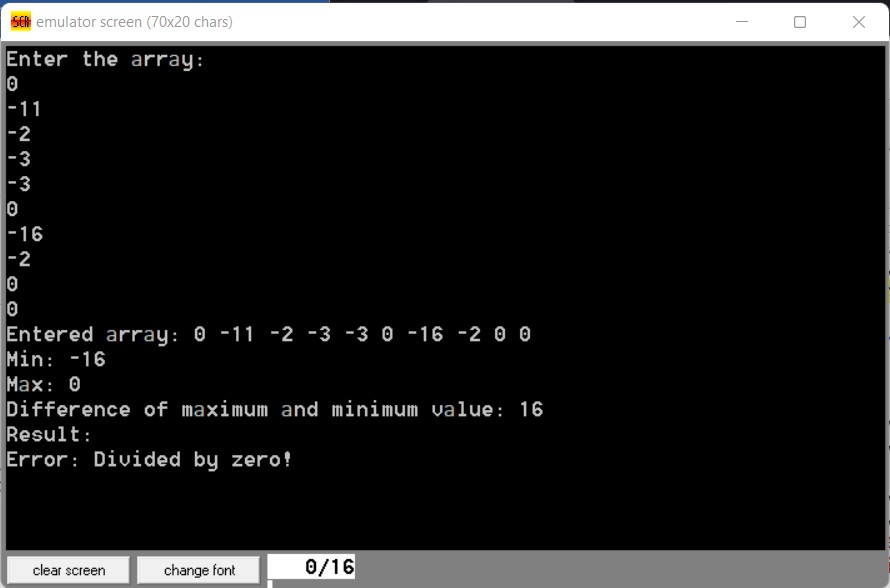
*call Exit*

*end start*

**Результат работы:**

****

****

****

**Вывод**

В ходе выполнения лабораторной работы было проведено ознакомление с арифметическими операциями над целочисленными данными, обработкой массивов чисел и правилами оформления ассемблерных процедур.