Министерство образования Республики Беларусь

Учреждение Образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Лабораторная работа №3

по дисциплине «Конструирование программ и языки программирования»

на тему «Целочисленные арифметические операции. Обработка массивов числовых данных»

вариант №6

|  |  |  |
| --- | --- | --- |
| Выполнил  студент гр. 250505  Зорин А.Ю. |  | Проверил  Туровец Н.О. |

Минск 2023

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

**Теоретические сведения**

Для выполнения работы требуется рассмотреть следующие элементы языка ассемблера:

*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. При умножении 8-битовых операндов результат всегда помещается в регистр AX. При умножении 16-битовых данных результат, который может быть длиною до 32 бит, помещается в пару регистров: в регистре DX содержатся старшие 16-бит, а в регистре AX – младшие 16-бит. Если старшая половина результата (АН, DX, EDX) содержит только нули (результат целиком поместился в младшую половину), то флаги CF и OF устанавливаются в 0, иначе – в 1. Значение остальных флагов (SF, ZF, AF и PF) не определено.

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).

Во всех трех вариантах считается, что результат может занимать в два раза больше места, чем размер источника. В первом случае приемник автоматически оказывается достаточно большим, но во втором и третьем случаях могут произойти переполнение и потеря старших бит результата. Флаги OF и CF будут равны единице, если это произошло, и нулю, если результат умножения 20 поместился целиком в приемник (во втором и третьем случаях) или в младшую половину приемника (в первом случае). Значения флагов SF, ZF, AF и PF после команды IMUL не определены.

-- 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.

Значения флагов CF, OF, SF, ZF, AF и PF после этой команды не определены, а переполнение (если частное больше того, что может быть помещено в регистр результата (255 для байтового деления и 65535 для деления слов)) или деление на ноль вызывает прерывание 0h.

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 XOR часто используется для обнуления регистра: XOR AX, AX ; обнуление AX

-- инверсия (NOT):

NOT приемник

Команда NOT выполняет инверсию бит приемника (регистр или переменная). 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.

Параметры в процедуры можно передавать в регистрах, в глобальных переменных, в стеке, в потоке кода, в блоке параметров. Одна из простых передач – передача параметров через регистры:

mov ax,word ptr value ; сделать копию значения

call procedure ; вызвать процедуру

**Код программы (.exe)**

.model small

.stack 100h

.data

mes0 db "Input chislo:",0Dh,0Ah,'$'

mes1 db "Input a massive:",0Dh,0Ah,'$'

mes2 db "Your massive:",0Dh,0Ah,'$'

mes3 db "Average value(Integer part): ",'$'

mes4 db "Average value(Fractional part): ",'$'

CrLf db 0Dh,0Ah,'$'

mas dw Len DUP (?)

averageInt dw 67

averageFract dw 67

fl dw ?

Len EQU 30

.code

Output macro mes

push ax

mov dx,offset mes

mov ah,09h

int 21h

pop ax

endm

ReadChar proc

mov ah,01h

int 21h

ret

ReadChar endp

ReadInteger proc

push cx

push bx

push dx

mov fl,0

xor cx, cx

mov bx, 10

call ReadChar

cmp al,'-'

je nnn

jmp nn

nnn:

mov fl,1

read:

call ReadChar

nn: cmp al, 13

je done

sub al, '0'

xor ah, ah

xor dx, dx

xchg cx, ax

mul bx

add ax, cx

xchg ax, cx

jmp read

done:

xchg ax, cx

cmp fl,1

je eee

jmp ee

eee:

neg ax

ee:

Output CrLf

pop dx

pop bx

pop cx

ret

ReadInteger endp

WriteChar proc

push ax

push dx

mov dl, al

mov ah, 2

int 21h

pop dx

pop ax

ret

WriteChar endp

WriteInteger proc near

push ax

push cx

push bx

push dx

xor cx, cx

mov bx, 10

cmp ax,0

jl ddd

jmp divl

ddd:

push ax

mov dl, '-'

mov ah, 2

int 21h

pop ax

neg ax

divl:

xor dx, dx

idiv bx

push dx

inc cx

cmp ax,0

jg divl

popl:

pop ax

add al, '0'

call WriteChar

loop popl

pop dx

pop bx

pop cx

pop ax

ret

WriteInteger endp

start:

mov ax,@data

mov ds,ax

Output mes1

mov di,0

mov cx,Len

input:

call ReadInteger

mov mas[di],ax

add di,2

loop input

xor ax,ax

xor di,di

mov cx,Len

count:

add ax,mas[di]

add di,2

loop count

mov bx,Len

cmp ax,0

jl d

jmp dd

d:

mov fl,1

neg ax

dd:

xor dx,dx

div bx

cmp fl,1

je z

jmp zz

z:

neg ax

zz:

mov averageInt,ax

mov averageFract,dx

mov ax,averageInt

Output mes3

call WriteInteger

Output CrLf

mov ax, averageFract

Output mes4

call WriteInteger

Output CrLf

mov ax,4C00h

int 21h

end start

**Вывод программы**

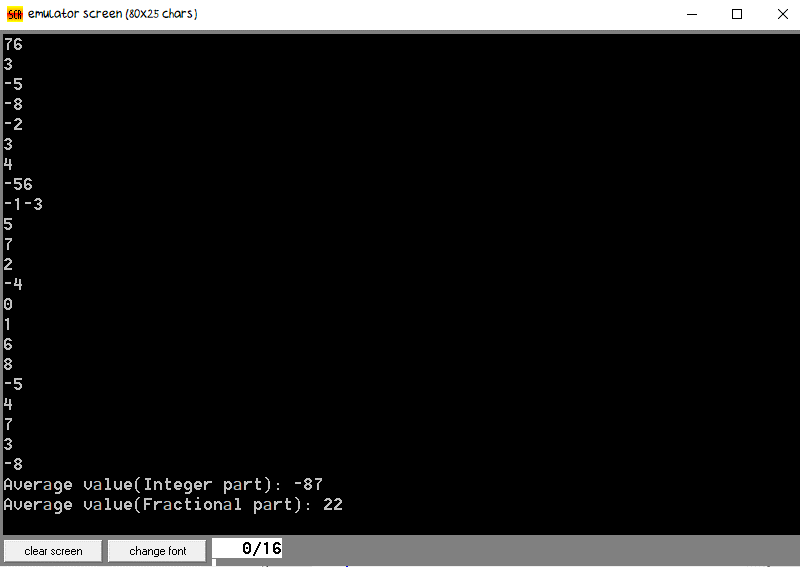


Рисунок 1 – Результат работы программы