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 «Программирование на языке ассемблера»

на тему «Создание простой программы на языке ассемблера»

вариант №1

|  |  |  |
| --- | --- | --- |
| Выполнил  студент гр.250502  Бекетова М.А. |  | 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

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 со-ответственно. 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 перемен-ная) и помещает результат в AL, АХ или ЕАХ, а остаток — в АН, DX или EDX со-ответственно. 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 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. Та-кая 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

При передаче параметров в стеке, для чтения параметров из стека в процедуре обычно используют не команду POP, а регистр ВР, в который помещают адрес вершины стека после входа в процедуру.

Для удобства ссылок на параметры, переданные в стеке, внутри функции иногда используют директивы EQU, чтобы не писать каждый раз точное смещение параметра от начала активационной записи (то есть от ВР).

**Код программы (.exe)**

.model small

.stack 100h

.data

entrInBsStr db "Enter initial base (2-16, DECIMAL):", 0Dh, 0Ah, '$'

entrInNumStr db 0Dh, 0Ah, "Enter initial number:", 0Dh, 0Ah, "(+/- 65535 (dec) is max value for a correct result)", 0Dh, 0Ah, '$'

entrNewBsStr db 0Dh, 0Ah, "Enter new base (2-16):", 0Dh, 0Ah, '$'

trnslStr db 0Dh, 0Ah, "Translated number:", 0Dh, 0Ah, '$'

overStr db 0Dh, 0Ah, "Overflow happened...", 0Dh, 0Ah, '$'

illStr db 0Dh, 0Ah, "Illegal characters...", 0Dh, 0Ah, '$'

wrongBsStr db 0Dh, 0Ah, "Illegal base...", 0Dh, 0Ah, '$'

wrongNumStr db 0Dh, 0Ah, "Number does not suit the base...", 0Dh, 0Ah, '$'

errStr db "The program has been terminated.", 0Dh, 0Ah, '$'

emptyStr db 0Dh, 0Ah, "Nothing has been entered...", 0Dh, 0Ah, '$'

simBssStr db 0Dh, 0Ah, "Bases are equal, there is no need in translation...", 0Dh, 0Ah, '$'

tmp db 200,?,200 dup('$')

inNum dw 0h

inBase dw 0h

finBase dw 0h

finNum dw 0h

isNegative dw 0h

.code

display macro string

lea dx, string

mov ah, 09h

int 21h

endm

input macro string

lea dx, string

mov ah, 0Ah

int 21h

endm

main:

;prepare ds

mov ax, @data

mov ds, ax

display entrInBsStr

input tmp

xor ax, ax ; clean reg and set 0

mov al,tmp[1] ; size

cmp al,0

je empty

push ax ; ax to stack

push 10 ; 10 to stack

lea di, tmp+2 ; adress of tmp+2 and save in di

call atoi ; char to int and save in ax dx

jo overflow ;!atoi

cmp dx, 0xFFFFh

je illChar ;==

cmp dx, 0xEEEEh

je illNum ;==

cmp dx, 1h

je illBase ;==

mov inBase, ax ; inbase == ax

push inBase ; inbase to stack

call checkBase

cmp dx, 0h ; if base <2, >16

je illBase

display entrInNumStr

input tmp

xor ax, ax

mov al,tmp[1]

cmp al,0

je empty

push ax

push inBase

lea di, tmp+2

call atoi

jo overflow

cmp dx, 0xFFFFh

je illChar

cmp dx, 0xEEEEh

je illNum

mov inNum, ax

cmp dx, 1h ; is the number negative

jne signConfirmed

inc isNegative

signConfirmed:

display entrNewBsStr

input tmp

xor ax, ax

mov al,tmp[1]

cmp al,0

je empty

push ax

push 10

lea di, tmp+2

call atoi

jo overflow

cmp dx, 0xFFFFh

je illChar

cmp dx, 0xEEEEh

je illNum

cmp dx, 1h

je illBase

mov finBase, ax ;final base

push finBase

call checkBase

cmp dx, 0h

je illBase

;if bases are simmilar

mov ax, inBase

cmp ax, finBase

je simBases

push inNum

push finBase

lea di, tmp+2

call translate

display trnslStr

push isNegative

push finBase

call printInRevOrder

jmp end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;errors;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

illChar:

display illStr

jmp error

illBase:

display wrongBsStr

jmp error

illNum:

display wrongNumStr

jmp error

overflow:

display overStr

jmp error

simBases:

display simBssStr

jmp end

empty:

display emptyStr

jmp end

error:

display errStr

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

end:

mov ax, 4c00h

int 21h

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;functions;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

atoi proc near

;stack contains base

push bp

mov bp, sp

mov bx, [bp+4]

mov cx, ax ;amount

xor dx, dx

sign dw 0h

tmpVal dw 0h

mov sign, dx ;to clear sign field

mov tmpVal, dx ;to clear tmp field

;sign check

mov dl, [di] ;entered number

mov ax, dx

cmp ax, '+'

je signConf

cmp ax, '-'

jne start

inc sign

signConf:

inc di

dec cx

start:

xor ax, ax

xor dx, dx

next:

mul bx ;

jo finish1

mov dl, [di]

mov tmpVal, ax

mov ax, dx

lowCheck:

cmp al, 'f'

ja charLocated

cmp al, 'a'

jb highCheck

sub dl, 57h

jmp checksPassed

highCheck:

cmp al, 'F'

ja charLocated

cmp al, 'A'

jb numCheck

sub dl, 37h

jmp checksPassed

numCheck:

cmp al, '9'

ja charLocated

cmp al, '0'

jb charLocated

sub dl, 30h

checksPassed:

cmp bl, dl ; cmp base and int

jbe wrongBase

possOvflwCheck:

mov ax, tmpVal

cmp ax, 0xFFF1h

jbe noOverflow

add ax, dx

cmp ax, 3h

ja noOverflow

;to catch overflow flag

mov ax, tmpVal

mul bx

jo finish1

noOverflow:

mov ax, tmpVal

add ax, dx

jo finish1

inc di

loop next

mov dx, sign

jmp finish1

wrongBase:

mov dx, 0xEEEEh

jmp finish1

charLocated:

mov dx, 0xFFFFh

finish1:

pop bp

ret

atoi endp

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

checkBase proc near

;stack contains: base for checking

push bp

mov bp, sp

mov bx, [bp+4]

mov ax, 2

cmp bx, ax

jl wrong2

mov ax, 16

cmp bx, ax

ja wrong2

mov dx, 0x1h ;flag if base not ok

jmp finish2

wrong2:

xor dx, dx

finish2:

pop bp

ret

checkBase endp

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

translate proc near

; stack contains: new base, initial number

push bp

mov bp, sp

mov bx, [bp+4] ;base

mov ax, [bp+6] ;number

xor cx, cx

dec di

again:

inc di

xor dx, dx

div bx ;num\newBase

mov [di], dl ;ostatok from div

inc cx

cmp ax, 0

ja again

pop bp

ret

translate endp

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

printInRevOrder proc near

;stack contains: base, sign

push bp

mov bp, sp

mov bx, [bp+4]

mov ax, [bp+6]

cmp ax, 0h

je nexta

mov dx, '-'

inc di

inc cx

jmp lowerThan9

nexta:

mov dl, [di]

add dx, 30h

cmp dx, 39h

jle lowerThan9

add dx,7

lowerThan9:

mov ah, 2

int 21h

dec di

loop nexta

pop bp

ret

printInRevOrder endp

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

end main

**Вывод программы**

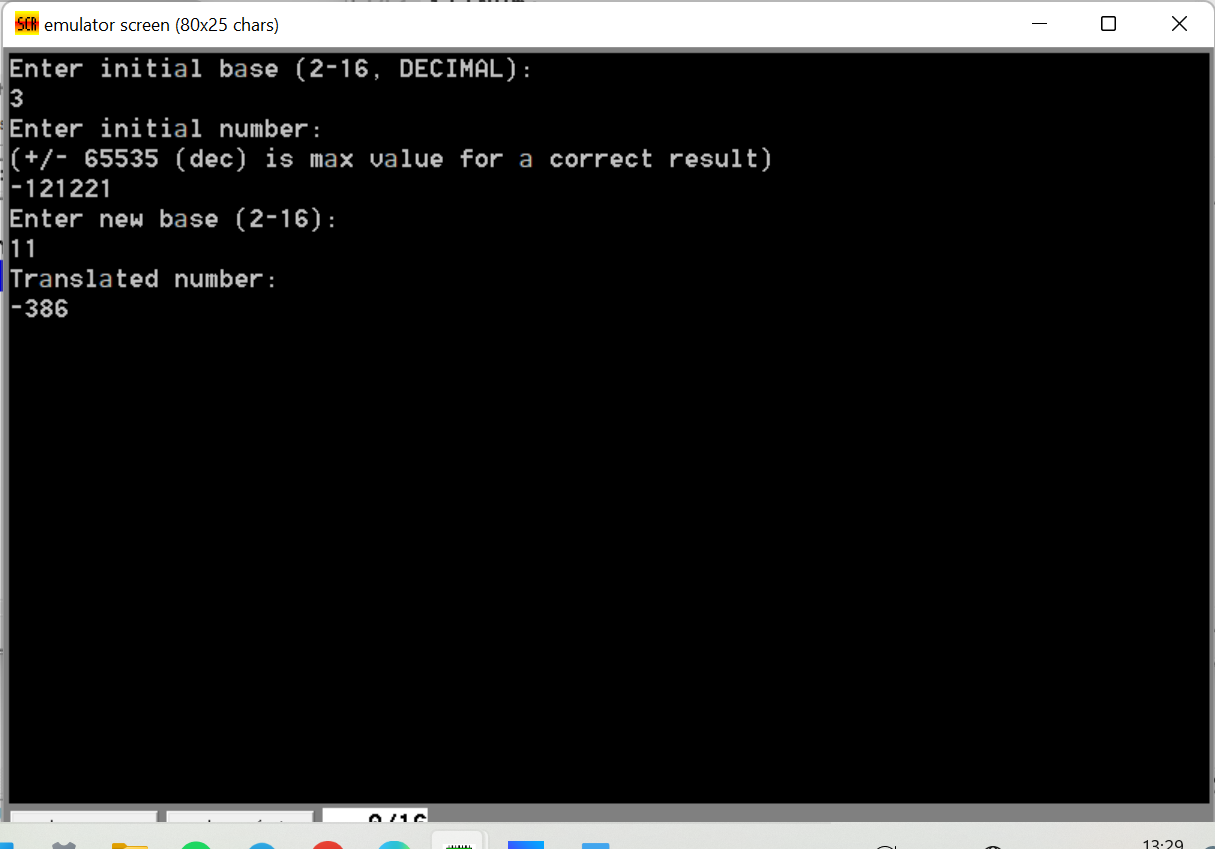


Рисунок 1 – Результат работы программы