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»

Вариант 2

Выполнил студент гр. 150502: Альхимович Н.Г.

Проверил:        Туровец Н.О.

Минск 2022

Цель работы:

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

Вариант задания:

Выполнить набор арифметических операций над двумя целыми числами, представленными в 10-ной системе счисления.

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 источ- ника, помещает сумму в приемник, не изменяя содержимое источника. Прием- ник может быть регистром или переменной, источник может быть числом, ре- гистром 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 ре- зультата), 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 соответ- ственно. 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 содержат- ся 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. Зна- чение 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 перемен- ная) 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 прием- ник (регистр).

Во всех трех вариантах считается, что результат может занимать в два ра- за больше места, чем размер источника. В первом случае приемник автоматиче- ски оказывается достаточно большим, но во втором и третьем случаях могут произойти переполнение и потеря старших бит результата. Флаги OF и CF бу- дут 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 перемен- ная) и помещает результат в AL, АХ или ЕАХ, а остаток — в АН, DX или EDX со- ответственно. Результат всегда округляется в сторону нуля, абсолютное значе- ние 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 опреде- лены, 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.

Ниже приведен фрагмент программы, выполняющей вычисление форму- лы X=1–(A\*2+B\*C)/(D–3) , где параметры X, A, B, C и D – 16-ти бито- вые целые знаковые числа:

...

MOV AX,2

IMUL A

MOV BX,DX

MOV CX,AX

MOV AX,B

IMUL C

ADD AX,CX

ADC DX,BX

MOV CX,D

SUB CX,3

IDIV CX

NEG AX

INC AX

; AX = 2 ;DX:AX=A\* 2

;BX:CX=A\* 2 ;DX:AX=B\* C ;DX:AX=A\* 2 +B\*C

;CX=D-3  
; AX = (A \* 2 + B \* C)/(D - 3) ;AX=-AX  
;AX=AX+1

MOV X,AX ; X = результат

...

2. Логические побитовые операции над целыми числами.

Логические побитовые операции над целыми числами выполняются с по- мощью 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 резуль- тат в приемник. Источник и приемник не могут быть переменными одновре- менно. Флаги OF и CF обнуляются, SF, ZF и PF устанавливаются в соответ- ствии 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 одно- временно. Флаги OF и CF обнуляются, SF, ZF и PF устанавливаются в соответ- ствии 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 резуль- тат в приемник. Источник и приемник не могут быть переменными одновре- менно. 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. Та- кая свобода выражения легко может приводить к трудночитаемым программам, и в язык ассемблера были включены директивы логического оформления про цедур:

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 коман-

ды 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, опре- деляя взаимодействие процедуры с языками высокого уровня. В некоторых ас- семблерах 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 пара- метров.

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

Параметры в процедуры можно передавать в регистрах, в глобальных пе- ременных, 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 про- цедуре обычно используют не команду POP, а регистр ВР, в который помещают адрес вершины стека после входа в процедуру:

push parameter1 ; поместить параметр 1 в стек

push parameter2 ; поместить параметр 2 в стек

call procedure

add sp,4

...

procedure proc near

push bp

mov bp,sp

...

mov ax,[bp+4]

; сохранить BP

; BP = вершина стека

; команды, которые могут использовать стек

; считать параметр 2

; освободить стек от параметров

; его адрес в сегменте стека ВР+4, потому что при выполнении

; команды CALL в стек поместили адрес возврата - 2 байта

; для процедуры типа NEAR (или 4 - для FAR), а потом еще и

; ВР - 2 байта

mov bx,[bp+6] ; считать параметр 1

... рор bp ret

procedure endp

; остальные команды

; восстановить BP

; возврат из процедуры

Для удобства ссылок на параметры, переданные в стеке, внутри функции иногда используют директивы EQU, чтобы не писать каждый раз точное сме- щение параметра от начала активационной записи (то есть от ВР), например, так:

push X

push Y

push Z

call procedure

...

procedure proc near

param\_z equ [bp+8]

param\_y equ [bp+6]

param\_x equ [bp+4]

push bp

mov bp,sp

... ; команды, которые могут использовать стек

mov ax,param\_x ; считать параметр X

... ; остальные команды процедуры

pop bp

ret 6 ; возврат с очисткой стека в процедуре

func endp

Код программы:

.model small

.stack 100h

.data

enter1 db "Enter the first number: ", '$'

enter2 db "Enter the second number: ", '$'

indent db 0ah, 0dh, '$'

res\_sum db "The sum: ", '$'

res\_sub db "The remainder: ", '$'

res\_mul db "The product: ", '$'

res\_div db "The quotient: ", '$'

over db "Overflow", '$'

max1\_len db 4

length1 db 0 ;actual length

num1 db 5 dup('$')

max2\_len db 4

length2 db 0

num2 db 5 dup('$')

ten dw 10

a dw ?

b dw ?

.code

start:

mov ax, @data

mov ds, ax

input1:

lea dx, enter1

mov ah, 09h

int 21h

mov ah, 0ah ;input of the string

lea dx, max1\_len

int 21h

lea dx, indent ;moving to the next line

mov ah, 09h

int 21h

lea si, num1 ;beginning of the string

mov bl, length1 ;length of the string

string1\_to\_int:

xor cx, cx ;preparing cx for getting the converted number

call stri

mov a, cx

input2:

lea dx, enter2

mov ah, 09h

int 21h

mov ah, 0ah

lea dx, max2\_len

int 21h

lea dx, indent

mov ah, 09h

int 21h

lea si, num2

mov bl, length2

lea dx, indent

mov ah, 09h

int 21h

string2\_to\_int:

xor cx, cx

call stri

mov b, cx

op\_sum:

lea dx, res\_sum

mov ah, 09h

int 21h

xor cx, cx

add cx, a

add cx, b

jc overflow

mov ax, cx

call print

lea dx, indent

mov ah, 09h

int 21h

op\_sub:

lea dx, res\_sub

mov ah, 09h

int 21h

xor cx, cx

add cx, a

sub cx, b

jc overflow

mov ax, cx

call print

lea dx, indent

mov ah, 09h

int 21h

op\_mul:

lea dx, res\_mul

mov ah, 09h

int 21h

xor ax, ax

add ax, a

mul b

jc overflow

call print

lea dx, indent

mov ah, 09h

int 21h

op\_div:

lea dx, res\_div

mov ah, 09h

int 21h

xor ax, ax

xor dx, dx

add ax, a

mov cx, b

div b

jc overflow

call print

lea dx, indent

mov ah, 09h

int 21h

exit:

mov ax, 4c00h

int 21h

ret

overflow:

lea dx, over

mov ah, 09h

int 21h

jmp exit

stri proc near

push ax ;preserve working registers

push bx

push dx

push si

next\_char:

cmp [si], '$' ;if the end of the string is reached

je done

cmp [si], '0' ;if char isn't a digit

jb done

cmp [si], '9'

ja done

mov bl, [si]

mov ax, cx ;ex. 545. 0->cx; cx+=5

mul ten ;5\*10->cx; cx+=4

mov cx, ax ;54\*10->cx; cx+=5

sub bl, 30h ;converting char to int

xor bh, bh

add cx, bx ;converted number is stored in cx

inc si ;moving to the next char

jmp next\_char

done:

pop si ;recover saved registers

pop dx

pop bx

pop ax

ret

stri endp

print proc near

cmp ax, 0 ;if a quotient is 0

jne not\_zero

add al, '0' ;printing '0' out

mov ah, 0eh

int 10h

ret

not\_zero:

push ax

push bx

push cx

push dx

mov bx, 10000

begin:

cmp bx, 0 ;if devider equals zero

jz end

cmp cx, 0 ;avoid printing zeros before numbers

je calc

cmp ax, bx ;if ax<bx, the result is 0

jb skip

calc:

xor cx, cx

xor dx, dx ;remainder

div bx

add al, 30h ;printing the last digit

mov ah, 0eh

int 10h

mov ax, dx ;remainder is stored in ax

skip: ;bx/=10

push ax

xor dx, dx

mov ax, bx

div ten

mov bx, ax

pop ax

jmp begin

end:

pop dx

pop cx

pop bx

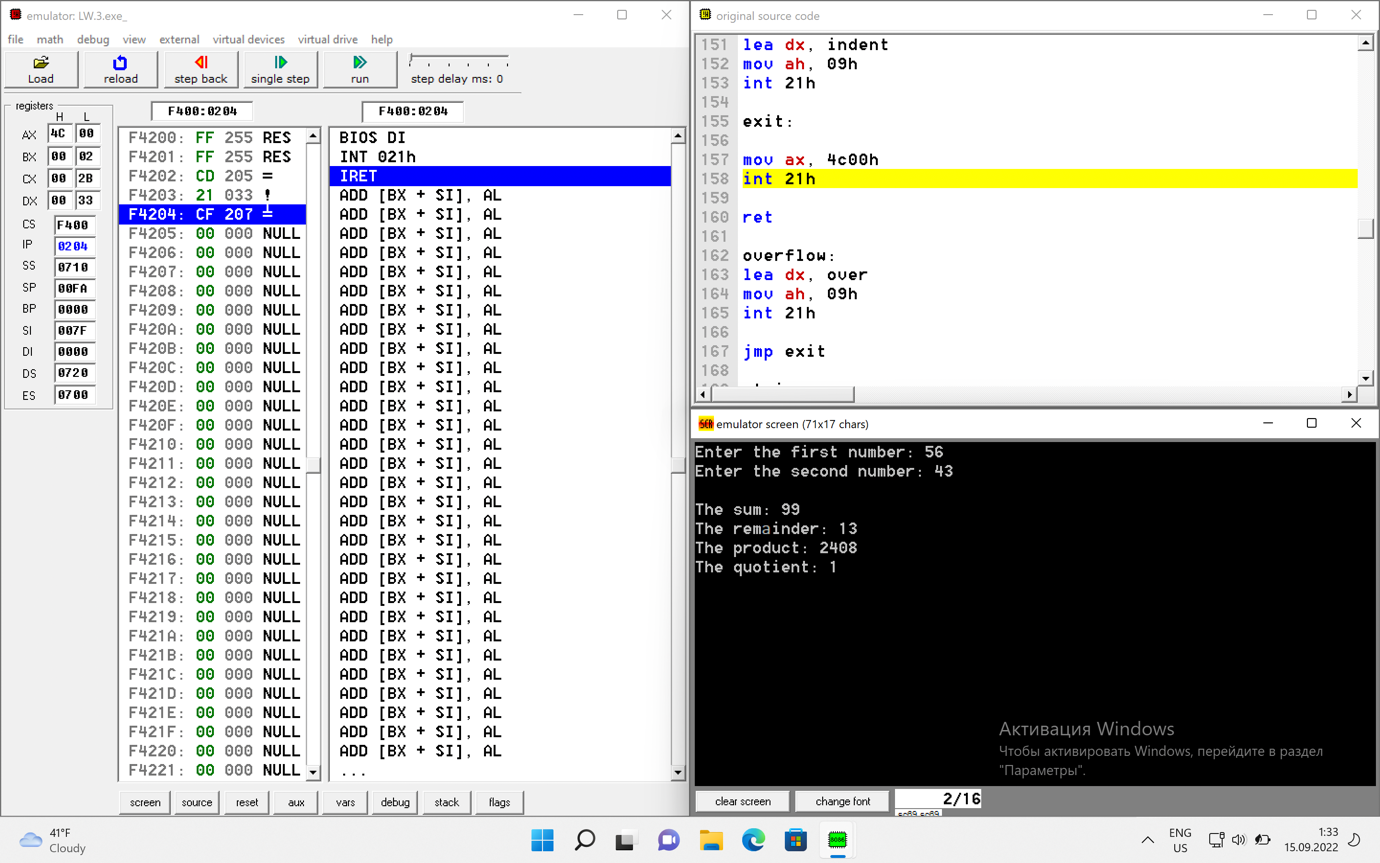
pop ax

ret

print endp

end start

Примеры работы программы:



Вывод:

В ходе лабораторной работы было написано приложение, выполняющее арифметические операции над целыми десятичными числами, введенными с клавиатуры. Я ознакомилась с арифметическими операциями над целочисленными данными, обработкой массивов чисел, а также с правилами оформления ассемблерных процедур.