Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

# **ОТЧЕТ**

По лабораторной работе №5

«Целочисленные арифметические операции. Обработка массивов числовых данных»

Дисциплина «Конструирование программ»

|  |  |  |
| --- | --- | --- |
| Выполнил: | студент группы 253502 | Р.Я. Ахметов |
| Проверил: | ассистент кафедры информатики | Н.С. Ячин |

Минск 2023

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

**Листинг программы и результаты ее работы:**

Вариант 3. Выполнить набор логических побитовых операций над двумя целыми числами, представленными в 10-ной системе счисления.

data segment

tmp\_str db 5, 0, 5 dup ('$')

num\_str db 5, 0, 5 dup ('$')

numstr db 10 dup ('$')

num dw 0

numa dw 0

numb dw 0

zero\_c db '0'

nine\_c db '9'

inpf\_str db "input first number$"

inps\_str db "input second number$"

inp\_str db "Input what to do:", 0Dh, 0Ah, " 1 - a and b", 0Dh, 0Ah, " 2 - a or b", 0Dh, 0Ah, " 3 - a xor b", 0Dh, 0Ah, " 4 - not a", 0Dh, 0Ah, " 5 - exit$"

out\_str db "ans: $"

i db 0

pkey db "press any key...$"

incn db "incorrect number, try one more time$"

ok\_str db "OK$"

clrf db 0Dh, 0Ah, '$'

ends

stack segment

dw 128 dup(0)

ends

code segment

; prtint new line

print\_nl macro

PUSHA

lea dx, clrf

mov ah, 9h

int 21h

POPA

endm

; prtint a string

print\_str macro str

PUSHA

lea dx, str

mov ah, 9

int 21h

lea dx, clrf

int 21h

POPA

endm

reverse\_str macro sii

LOCAL find\_end, rev, done

PUSHA

mov si, sii

mov di, si

find\_end:

lodsb

cmp al, '$'

jne find\_end

sub si, 2

rev:

cmp di, si

jge done

mov cl, [di]

mov al, [si]

mov [si], cl

mov [di], al

dec si

inc di

jmp rev

done:

POPA

endm

num\_to\_str macro num, str

LOCAL loopa, p4, ne

PUSHA

mov ax, num

lea si, str

loopa:

cmp ax, 0

jge p4

mov [si], '-'

inc si

neg ax

p4:

mov cx, 10

div cl

mov bh, ah

add bh, 30h

mov [si], bh

inc si

xor ah, ah

cmp al, 0

jne loopa

mov [si], '$'

lea si, str

cmp [si], '-'

jne ne

inc si

ne:

reverse\_str si

POPA

endm

str\_to\_num MACRO str, result

LOCAL pl, ng, done, enda, pq

PUSHA

mov ax, 0

mov bx, 0 ; Clear the result register

mov cx, 10 ; Set the base to 10

lea si, str ; Point SI to the string buffer

mov bl, [si]

cmp bl, '-' ; Check if the number is negative

je ng

pl:

cmp bl, '$' ; Check for the null terminator

je done

sub bl, '0' ; Convert character to digit

mul cx ; Multiply result by 10

add ax, bx ; Add the digit to the result

inc si ; Move to the next character

mov bl, [si] ; Load the next character

jmp pl

ng:

inc si

mov bl, [si] ; Skip the negative sign

jmp pl

pq:

neg ax

jmp enda

done:

lea si, str

mov bl, [si]

cmp bl, '-'

je pq

enda:

mov result, ax

POPA

ENDM

input\_str macro str

PUSHA

lea dx, str

mov ah, 0Ah

int 21h

POPA

endm

check\_num macro c\_num

LOCAL nn, an, exit

PUSHA

mov cx, 1

cmp zero\_c, c\_num

jg nn

cmp nine\_c, c\_num

jb nn

jmp an

nn:

cmp cx, 0

jmp exit

an:

cmp cx, 1

exit:

POPA

endm

input\_num macro tmpstr, numstr

LOCAL tryAgain, start, lup, lupa, ne, endStr, exit

PUSHA

jmp start

tryAgain:

print\_nl

print\_str incn

start:

lea ax, num

input\_str tmpstr

lea si, tmpstr

cld

mov cx, si[1]

cmp si[2], '-'

jne ne

inc si

ne:

add si, 2

lup:

cmp [si], 0Dh

je endStr

mov al, [si]

check\_num al

jne tryAgain

inc si

loop lup

endStr:

mov [si], '$'

lea si, tmpstr

lea di, numstr

lupa:

mov al, si[2]

mov [di], al

cmp si[2], '$'

je exit

inc si

inc di

jmp lupa

exit:

POPA

endm

program\_end macro

lea dx, pkey

mov ah, 9

int 21h ; output string at ds:dx

mov ah, 1

int 21h

mov ax, 4c00h ; exit to operating system.

int 21h

endm

start:

mov ax, data

mov ds, ax

mov es, ax

print\_str inpf\_str

input\_num tmp\_str numstr

str\_to\_num numstr num

mov ax, num

mov numa, ax

print\_nl

print\_str inps\_str

input\_num tmp\_str numstr

str\_to\_num numstr num

mov bx, num

mov numb, bx

p1:

print\_nl

print\_str inp\_str

print\_nl

input\_num tmp\_str numstr

print\_nl

str\_to\_num numstr num

mov cx, num

cmp cx, 0

jl p1

cmp cx, 5

jg p1

cmp cx, 1

je j1

cmp cx, 2

je j2

cmp cx, 3

je j3

cmp cx, 4

je j4

jmp exit

j1:

mov ax, numa

and ax, bx

num\_to\_str ax numstr

print\_str out\_str

print\_str numstr

print\_nl

jmp p1

j2:

mov ax, numa

or ax, bx

num\_to\_str ax numstr

print\_str out\_str

print\_str numstr

print\_nl

jmp p1

j3:

mov ax, numa

xor ax, bx

num\_to\_str ax numstr

print\_str out\_str

print\_str numstr

print\_nl

jmp p1

j4:

mov ax, numa

not ax

num\_to\_str ax numstr

print\_str out\_str

print\_str numstr

print\_nl

jmp p1

exit:

program\_end

ends

end start

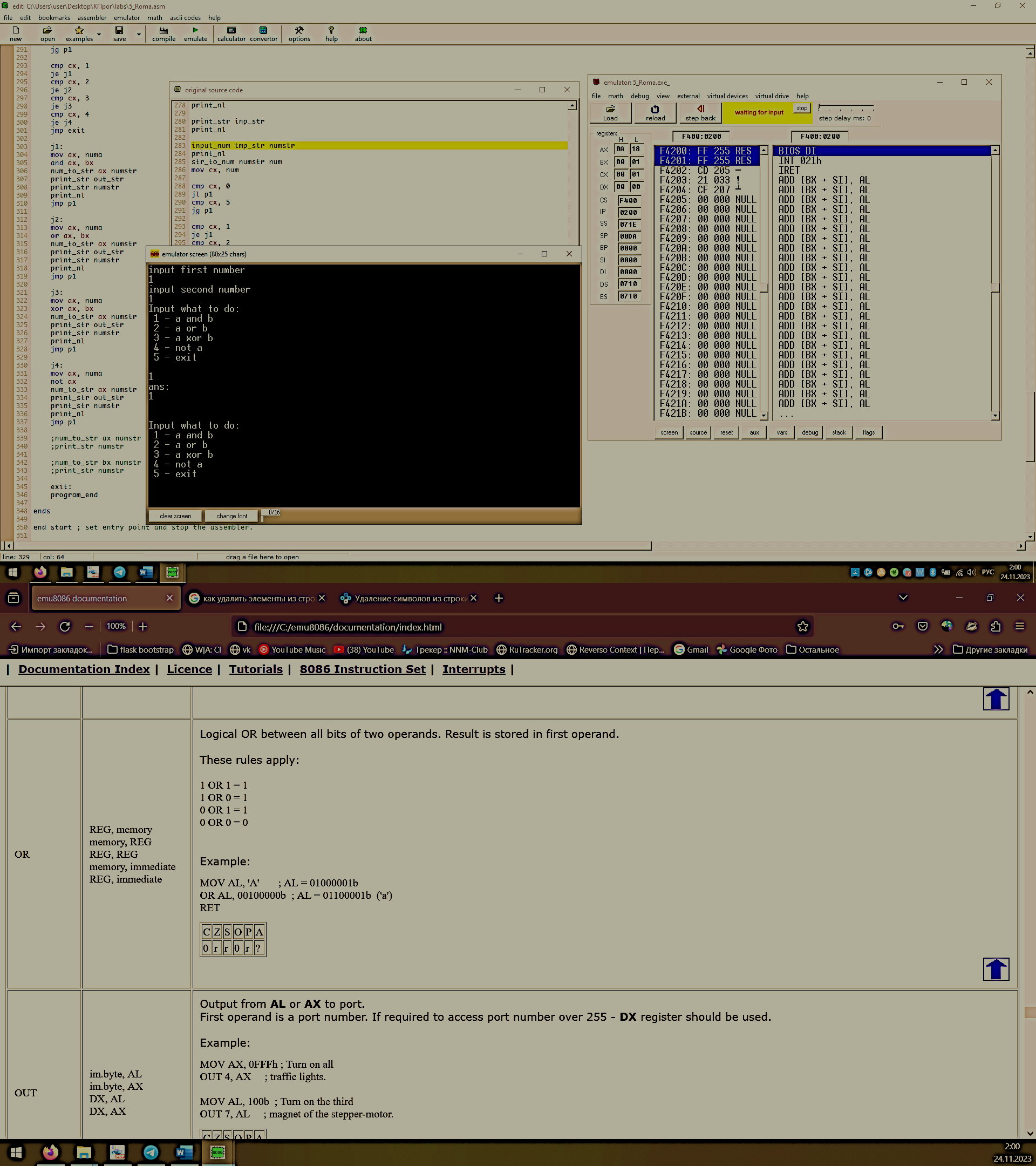


Рисунок 1.1 – Результат выполнения программы

**Контрольные вопросы:**

1. Процедуры в ассемблере.  
    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. Оформление процедуры:  
    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 определяет 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.  
    Способы передачи параметров в процедуры: в регистрах, в глобальных переменных, в стеке, в потоке кода, в блоке параметров.
2. Проблемы ввода чисел в ассемблерных программах (знак числа, переполнение при вводе).  
    Ассемблер не предоставляет средства для ввода чисел, поэтому числовые данные, считанные в символьном формате подлежат обязательной дополнительной обработке и проверке на переполнение и знак (для инвертирования знака используется команда NEG).
3. Арифметические операции над целыми числами.  
    Команды, выполняющие арифметические операции над целыми числами:  
    - команды сложения (ADD, ADC): ADD (ADC) приемник, источник  
    Команда ADD выполняет арифметическое сложение приемника и источника, помещает сумму в приемник, не изменяя содержимое источника. Нельзя использовать переменную одновременно и для источника, и для приемника.   
    Команда ADC во всем аналогична ADD, кроме того, что она выполняет арифметическое сложение приемника, источника и флага СF.  
    - команды вычитания (SUB, SBB): SUB приемник, источник  
    Команда SUB вычитает источник из приемника и помещает разность в приемник. Нельзя использовать переменную одновременно и для источника, и для приемника. SUB не делает различий между числами со знаком и без знака, но флаги позволяют использовать ее как для тех, так и для других.  
    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 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.  
    Команда IMUL выполняет умножение с учетом знака. Эта команда имеет три формы, различающиеся числом операндов.  
    - 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.  
    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.
4. Переполнение при арифметических операциях.  
    Для организации правильного процесса сложения, вычитания либо умножения чисел после использования соответствующих арифметических команд необходимо учитывать состояние флагов CF и OF. При беззнаковом сложении/вычитании флаг CF отражает перенос/заем из старшего разряда, при знаковом сложении/вычитании OF показывает, не вышел ли результат за диапазон представления знаковых чисел. После завершения операции умножения CF = 0 и OF = 0, если старшая часть результата нулевая; CF = 1 и OF = 1, если результат вышел за пределы младшей части произведения и состоит из двух частей. Для анализа состояния этих флагов можно использовать команды JC\JNC и JO\JNO.  
    При выполнении команд деления 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 вызывают прерывание 0h.
5. Логические побитовые операции над целыми числами.  
    Команды, выполняющие логические побитовые операции над целыми числами:  
    - логическое И (AND): AND приемник, источник;  
    - логическое ИЛИ (OR): OR приемник, источник;  
    - логическое исключающее ИЛИ (XOR): XOR приемник, источник;  
    - инверсия (NOT): NOT приемник.  
    После выполнения команд AND, OR, XOR флаги OF и CF обнуляются, SF, ZF и PF устанавливаются в соответствии с результатом, AF – не определен; команда NOT не затрагивает флаги.

**Вывод:** В состав процессора Intel 8086 входит набор арифметических операций, которые позволяют работать как с знаковыми, так и беззнаковыми числами, и корректно обрабатывать случаи переполнения и деления на ноль, а также логических побитовых операций над целыми числами. Язык ассемблера содержит довольно мощные средства поддержки модульного подхода в рамках структурного программирования. В языке ассемблера эта технология поддерживается в основном с помощью механизма процедур. Гибкость интерфейса между процедурами достигается за счет разнообразия вариантов передачи аргументов в процедуру и возвращения результатов. Для этого могут использоваться регистры, общие области памяти, стек.