Министерство науки и образования РФ

Федеральное государственное автономное образовательное

учреждение высшего профессионального образования

«Санкт-Петербургский государственный электротехнический

университет «ЛЭТИ» им. В. И. Ульянова (Ленина)»

(СПбГЭТУ «ЛЭТИ»)

Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

Отчёт

по лабораторной работе № 3

на тему:

“Способы адресации”

по дисциплине “Организация ЭВМ и Систем”

Вариант 5

Выполнил студент гр. 4306: Табаков А.В.

Принял: Манирагена Валенс

Санкт-Петербург  
2016

**Цель**

Ознакомиться со способами адресации данных на языке ассемблера intel 8086. Научиться работать с разными сегментами данных.

**Задание**

Задано два массива. Каждому элементу одного массива присвоить значение большего из одноименных элементов из обоих массивов. Массивы локализованы в разных сегментах с различными смещениями.

**Текст программы**

.Model tiny

.Data

greeting db "This programm print array with max elements of given two arrays", 0dh, 0ah, "$"

textSizeF db "Please input size of first array from 1 to 9", 0dh, 0ah,"first array size = $"

textSizeS db "Please input size of second array from 1 to 9", 0dh, 0ah,"second array size = $"

textNum db "Please input numbers from 0 to 9", 0dh, 0ah,"$"

textRes db "Result saved in RESARR$"

equSym db " = $"

endl db 0ah, 0dh, "$"

textFirstArr db "DSARR[$"

textSecondArr db "ESARR[$"

textResArr db "RESARR[$"

closeBracket db "]$"

question db "Press 0 for retry, any another to exit", 0dh, 0ah, "$"

pkey db "Press any key...$"

firstArr dw 10 dup(0)

firstArrSize dw ?

secondArr dw 10 dup(0)

secondArrSize dw ?

resArrSize dw ?

resArr dw 10 dup(0)

temp dw ?

buffer db 6 ;max num with 5 symbols

blength db ?

bconteg: ;consistance of buf is over of prog

hexstring equ bconteg

.Stack 0100h

.Code

start:

mov ax, @data

mov ds, ax

mov es, ax

call setDisp

lea dx, greeting ;greeting message

mov ah, 09h

int 21h

lea dx, textSizeF ;enter size of first arr msg

mov ah, 09h

int 21h

call input ;input size of first arr

mov firstArrSize, ax

mov temp, ax

inpFirstArr:

call endlp

lea dx, textFirstArr ;DSARR[

mov ah, 09h

int 21h

mov ax, firstArrSize

sub ax, temp

call printAX ;i for DSARR[i]

mov dl, ']'

mov ah, 02h

int 21h

mov dl, '='

mov ah, 02h

int 21h

call input ;write in DSARR[i] arr num

mov cx, ax

mov ax, firstArrSize

sub ax, temp

mov bx, 2h

mul bx

mov bx, ax

mov ds:[bx+firstArr], cx

dec temp

cmp temp, 0

jnz inpFirstArr

call endlp

lea dx, textSizeS ;enter size of first arr msg

mov ah, 09h

int 21h

call input ;input size of second arr

mov secondArrSize, ax

mov temp, ax

inpSecondArr:

call endlp

lea dx, textSecondArr ;ESARR[

mov ah, 09h

int 21h

mov ax, secondArrSize

sub ax, temp

call printAX ;i for ESARR[i]

mov dl, ']'

mov ah, 02h

int 21h

mov dl, '='

mov ah, 02h

int 21h

call input ;write in ESARR[i] arr num

mov cx, ax

mov ax, secondArrSize

sub ax, temp

mov bx, 2h

mul bx

mov bx, ax

mov es:[bx+secondArr], cx

dec temp

cmp temp, 0

jnz inpSecondArr

call endlp

;Main code

mov cx, firstArrSize

cmp cx, secondArrSize

jnb sizeOk

mov cx, secondArrSize

sizeOk:

mov resArrSize, cx ;resArrSize=max(firstArrSize, secondArrSize)

xor bx, bx ;bx = 0

mainL: ;resARR[bx] = max(DSARR[bx], ESARR[bx])

mov ax, ds:[firstArr+bx] ;ax = DSARR[bx]

cmp ax, es:[secondArr+bx] ;if ax < ESARR[bx]

jb mainElse

mov es:[resArr+bx], ax

jmp mainEnd

mainElse:

mov ax, es:[secondArr+bx]

mov es:[resArr+bx], ax

mainEnd:

add bx, 2

loop mainL

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

mov ax, resArrSize

mov temp, ax

lea dx, textRes ;Result saved in RESARR

mov ah, 09h

int 21h

printing:

call endlp

lea dx, textResArr ;RESARR[

mov ah, 09h

int 21h

mov ax, resArrSize

sub ax, temp

call printAX ;i for RESARR[i]

mov dl, ']'

mov ah, 02h

int 21h

mov dl, '='

mov ah, 02h

int 21h

mov cx, ax ;print RESARR[i]

mov ax, resArrSize

sub ax, temp

mov bx, 2h

mul bx

mov bx, ax

mov ax, es:[resArr+bx]

call printAX

dec temp

cmp temp, 0

jnz printing

call endlp

lea dx, question ;Enter 0 for retry, 1 to exit

mov ah, 09h

int 21h

mov ah, 01h

int 21h

cmp al, '0'

call start

call quit

proc setDisp

xor dx,dx ;cursor's position

mov ah,02h ;set at (0,0)

int 10h

mov bl,00001010b ;colors green on black

mov cx,25\*80 ;count of simbols on display

mov ax,0920h ;printing 25\*80 spaces

int 10h

ret

endp

proc quit

mov ax, 4c00h ; exit to operating system.

int 21h

endp

proc endlp ;press enter

push dx

push ax

lea dx, endl

mov ah, 09h

int 21h

pop ax

pop dx

ret

endp

proc input

lea dx,buffer ;buffer's address

mov ah,0ah ;write in buffer

int 21h

;from string to bin

xor di,di ;start of buffer

xor ax,ax ;clear ax

mov cl,blength

xor ch,ch

xor bx,bx

mov si,cx ;buffer's length

mov cl,10 ;multiplier

toHex:

mov bl,byte ptr bconteg[di]

sub bl,'0' ;num = num's code - 30h

jb badInp ;if symbol not a num

cmp bl,9 ;same

ja badInp ;try input again

mul cx ;multiply on 10

add ax,bx ;+new num to ax

inc di ;next symbol

cmp di,si ;if di<blength + 1

jb toHex

nM:

jmp endInp

badInp:

jmp start

endInp:

ret

endp

proc printAX

push cx

push bx

mov bx,0ah ;divider

xor cx,cx ;clear count

divloop:

xor dx,dx ;clear dx

div bx ;divide on 10

add dx,'0' ;make a symbol from num

push dx ;save dx

inc cx

test ax,ax ;if ax!=0

jnz divloop ;continue to divide

restore:

;pop ax

pop ax ;read from stack

mov dx, ax

mov ah,2 ;print symbol from al

int 21h ;

loop restore

pop bx

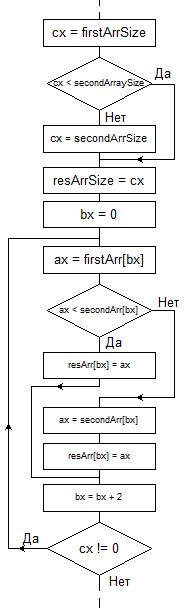
pop cx

ret

endp

end start ; set entry point and stop the assembler.

**Блок-схема основного алгоритма**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Трассировка основного алгоритма программы** | | | | |
| Адрес | Мнемокод | Двоичный код | Изменения данных | Комментарий |
| 00BC | mov  cx, firstArrSize | Байт 1: 10001011  100010 – операция переноса  1 – сначала приёмник потом источник  1 – слово (0 - 8 бит, 1 - 16 бит)  Байт 2: код cx и доп. информ.  Байт 3-4: адрес firstArrSize | cx = firstArrSize | Флаги установлены:  Нуля, Паритета. |
| 00C0 | cmp cx, secondArrSize | Байт 1: 00111011  001110 – операция сравнения  1 – сначала приёмник потом источник  1 – слово (0 - 8 бит, 1 - 16 бит)  Байт 2: код cx и доп. информ.  Байт 3-4: адрес secondArrSize |  | Устанавливается флаг нуля если, данные равны, если cx < SAS то устанавливается флаг переноса |
| 00C4 | jnb sizeOk | Байт 1: 01110100 – операция jnb  Байт 2: 00000100 – смещение |  | Если не установлен флаг переноса, то переходим |
| 00C6 | mov  cx, secondArrSize | Байт 1: 10001011  100010 – операция переноса  1 – сначала приёмник потом источник  1 – слово (0 - 8 бит, 1 - 16 бит)  Байт 2: код cx и доп. информ.  Байт 3-4: адрес secondArrSize | cx = secondArrSize |  |
| 00CA | mov resArrSize, cx | Байт 1: 10001001  100010 – операция переноса  0 – сначала источник потом приёмник  1 – слово (0 - 8 бит, 1 - 16 бит)  Байт 2: код cx и доп. информ.  Байт 3-4: адрес resArrSize | resArrSize = cx |  |
| 00CE | xor bx, bx | Байт 1: 00110011  001100 – операция XOR  1 – сначала приёмник потом источник  1 – слово (0 - 8 бит, 1 - 16 бит)  Байт 2: 11011011  11 – указываем что работаем с регистрами  011 – код регистра приёмника (BX)  011 – код регистра источника (BX) | bx = 0 | Флаги установлены:  Нуля, Паритета. |
| 00D0 | mov ax, ds:[firstArr+bx] | Байт 1: 10001011  100010 – операция переноса  1 – сначала приёмник потом источник  1 – слово (0 - 8 бит, 1 - 16 бит)  Байт 2: коды ax, bx и доп. информ.  Байт 3-4: адрес firstArr | ax = firstArr[bx] |  |
| 00D4 | cmp ax, es:[secondArr+bx] | Байт 1: 00100110 – сравнение регистра ax c переменной в памяти  Байт 2-3: код bx и адрес es  Байт 4-5: адрес secondArr |  | Устанавливается флаг нуля если, данные равны, если ax < secondArr[bx] то устанавливается флаг переноса |
| 00D9 | jb mainElse | Байт 1: 01110010 – операция jb  Байт 2: 00001100 – смещение |  | Если установлен флаг переноса, то переходим |
| 00DB | mov es:[resArr+bx], ax | Байт 1: 00100110 – перенос в массив в сегменте  Байт 2-3: код ax и адрес es + доп инф.  Байт 4-5: адрес resArr | resArr [bx] = ax |  |
| 00E0 | jmp mainEnd | Байт 1: 11101011 – операция jmp  Байт 2: 00001011 – смещение |  |  |
| 00E2 | nop | Байт 1: 10010000 |  | Зарезер. байт Компилятор. |
| 00E3 | mov ax, es:[secondArr+bx] | Байт 1: 10001011  100010 – операция переноса  1 – сначала приёмник потом источник  1 – слово (0 - 8 бит, 1 - 16 бит)  Байт 2: коды ax, bx и доп. информ.  Байт 3-4: адрес secondArr | ax = secondArr[bx] |  |
| 00E8 | mov es:[resArr+bx], ax | Байт 1: 00100110 – перенос в массив в сегменте  Байт 2-3: код ax, bx, адрес es + доп инф.  Байт 4-5: адрес resArr | resArr[bx] = ax |  |
| 00ED | add bx, 2 | Байт 1: 10000011 – сложение регистра с числом  Байт 2: код bx + доп. Инф.  Байт 3: число 2 | bx = bx+2 |  |
| 00F0 | loop mainL | Байт 1: 11100010 – операция loop  Байт 2: Адрес метки mainL |  |  |

Вывод

Я ознакомился со способами адресации данных на языке ассемблера intel 8086. Научился использовать разные сегменты данных.