1. Команды mov, add и sub не поддерживают формат выполнения типа память-память(mem-mem)!

В регистре EDX находится остаток от деления.

1. ***Команды «развертывания» чисел:***

**CBW** ; байт в слово AL -> AX

**CWD** ; слово в двойное слово AX -> DX:AX

**CDQ** ; двойное слово в учетверенное EAX -> EDX:EAX

**CWDE** ; слово в двойное слово AX -> EAX

1. Правильное деление с учетом знака

.DATA?

a sword ?

…

mov AX,a ;AX:=a

mov CX,2 ;CX:=2

cwd

idiv word PTR CX ;AX:=a/2

1. Ввод и вывод

.DATA

buffer DB 100 dup(?),0

…

Invoke StdIn, ADDR buffer,LengthOf buffer; ввод

Invoke StripLF, ADDR buffer ; символ конца строки меняем на ноль

Invoke atol, ADDR buffer ; преобразование строки в число, результат в EAX.

При выводе следует помнить, что используется регистр ECX, поэтому при обходе матрицы или массива вывод надо организовать следующим образом:

…

push ECX

Invoke StdOut, ADDR Your string

pop ECX

…

1. Ввод массива(матрицы) организуется следующим образом:

.DATA

buffer DB 100 dup(?),0

X sword 0,0,0,0,0

sword 0,0,0,0,0

sword 0,0,0,0,0

sword 0,0,0,0,0

sword 0,0,0,0,0

…

mov EBX,0 ; смещение

mov ECX,25 ; количество элементов массива

Enter\_cycle:

push ECX

Invoke StdOut, ADDR Enter\_Str ; приглашение к вводу

Invoke StdIn, ADDR buffer,LengthOf buffer ; ввод

Invoke StripLF, ADDR buffer ; меняем символ конца строки на ноль

Invoke atol, ADDR buffer ; преобразование в число, результат в EAX

mov X[EBX],AX ; помещаем то что ввели в X[EBX]

add EBX,2 ; смещение при типе word

pop ECX

loop Enter\_cycle ; цикл через loop вычитает из ECX единицу при каждой итерации

1. Обход и вывод матрицы

.DATA

Out\_Message DB "Output the matrix",13,10,0

Pusto DB 13,10,0

Probel DB ' ',0

…

Invoke StdOut,ADDR Out\_Message

mov EBX,0

mov ECX,5

…

Outcycle1:

…

push ECX

mov ECX,5

…

Outcycle2:

…

mov AX,X[EBX\*2]

push ECX

Invoke dwtoa,X[EBX\*2], ADDR buffer

Invoke StdOut, ADDR buffer

Invoke StdOut, ADDR Probel

pop ECX

inc EBX

loop Outcycle2

pop ECX

push ECX

Invoke StdOut, ADDR Pusto

pop ECX

loop Outcycle1

* *пересылка цепочки*:

movs адрес\_приемника,адрес\_источника

movsb

movsw

movsd

* *сравнение цепочек*:

cmps адрес\_приемника,адрес\_источника

cmpsb

cmpsw

cmpsd

* *сканирование цепочки*:

scas адрес\_приемника

scasb

scasw

scasd

* *загрузка элемента из цепочки*:

lods адрес\_источника

lodsb

lodsw

lodsd

* *сохранение элемента в цепочке*:

stos адрес\_приемника

stosb

stows

stosd

Префиксы повторения имеют свои мнемонические обозначения:

rep   
repeили repz

repne или repnz

Эти префиксы повторения указывают перед нужной цепочечной командой. Цепочечная команда без префикса выполняется один раз. Размещение префикса перед цепочечной командой заставляет ее выполняться в цикле.

Отличия приведенных префиксов в том, на каком основании принимается решение о циклическом выполнении цепочечной команды:

- *по состоянию регистра* ***ecx/cx***

*- по флагу нуля* ***zf*:**

**Цепочка-источник**, адресуемая операндом *адрес\_источника*, ***может***находиться в текущем сегменте данных, определяемом регистром **ds**.

**Цепочка-приемник**, адресуемая операндом *адрес\_приемника*, ***должна*** быть в дополнительном сегменте данных, адресуемом сегментным регистром **es**.

**cld** (Clear Direction Flag) — очистить флаг направления. Команда сбрасывает флаг направления **df** в 0.

**std** (Set Direction Flag) — установить флаг направления. Команда устанавливает флаг направления **df** в 1.

Написать программу пересылки строки символов из строки Source в строку Result.

; Template for console application

.586

.MODEL flat, stdcall

OPTION CASEMAP:NONE

Include kernel32.inc

Include masm32.inc

IncludeLib kernel32.lib

IncludeLib masm32.lib

.CONST

MsgExit DB 0AH,0DH,"Press Enter to Exit",0AH,0DH,0

**Source db 'Primer of mov string' ;** Исходная строка

.DATA

.DATA?

inbuf DB 100 DUP (?)

**Result db 22 DUP (' ')** ;Строка результат

.CODE

**Start:**

**cld** ; сброс флага DF — обработка строки от начала к концу

**lea esi,Source** ; загрузка в si смещения строки-источника

**lea edi,Result** ; загрузка в DS смещения строки-приёмника

**mov Ecx,20** ; для префикса rep — счетчик повторений (длина строки)

**rep movsb** ;пересылка строки источника в строку приемник

XOR EAX,EAX

XOR EAX,EAX

**Invoke StdOut,ADDR Result ;** вывод строки результата

Invoke StdOut,ADDR MsgExit

Invoke StdIn,ADDR inbuf,LengthOf inbuf

Invoke ExitProcess,0

**End Start**

|  |  |
| --- | --- |
| **Причина прекращения операции сравнения** | **Команда условного перехода, реализующая переход по этой причине** |
| операнд\_источник > операнд\_приемник | jg |
| операнд\_источник = операнд\_приемник | je |
| операнд\_источник <> операнд\_приемник | jne |
| операнд\_источник < операнд\_приемник | jl |
| операнд\_источник <= операнд\_приемник | jle |
| операнд\_источник >= операнд\_приемник | jge |

**Пример 2.** Написать программу проверки вхождения подстроки в строку символов. Строка и подстрока задаются константами в программе.

; Template for console application

.586

.MODEL flat, stdcall

OPTION CASEMAP:NONE

Include kernel32.inc

Include masm32.inc

IncludeLib kernel32.lib

IncludeLib masm32.lib

.CONST

MsgExit DB 0AH,0DH,"Press Enter to Exit",0AH,0DH,0

IsxStroka db 'Primer of string mov' ;исходная строка

PodStroka db 'string' ;подстрока

yes db 'yes' ;подстрока входит в строку

no db 'not' ;подстрока не входит в строку

.DATA

.DATA?

inbuf DB 100 DUP (?)

Result db 22 DUP (' ') ;строка для вывода результата

.CODE

Start:

**cld** ; сброс флага DF - обработка строки от начала к концу

**lea edi,IsxStroka** ; загрузка в *esi* смещения строки для проверки

**lea esi,PodStroka** ; загрузка в *edi* смещения подстроки для проверки

**mov ecx,**15; Количество проверок строки и подстроки ls-lp+1

**cycl: push edi**

**push esi**

**push ecx**

**mov Ecx,6**  ; для префикса rep - счетчик повторений (длина подстроки)

**repe cmpsb**

**jne next** ; Если сравниваемые символы различны, а сх=0

**jcxz equal** ;cx=0, и символы одинаковы, то все символы строки и подстроки совпадают

**next: pop ecx**

**pop esi**

**pop edi**

**inc edi**

**loop cycl**

**jmp not\_equa**l ; вышли по счетчику внешнего цикла, значит строки не совпадают

equal**:** mov ecx,3

lea esi,yes

lea edi,Result ; подготовка вывода сообщения YES совпадают

rep movsb

jmp kon

not\_equal: mov ecx,3

lea esi,no

lea edi,Result ; подготовка вывода сообщения NOT не совпадают

rep movsb

kon:

XOR EAX,EAX

Invoke StdOut,ADDR Result

Invoke StdOut,ADDR MsgExit

Invoke StdIn,ADDR inbuf,LengthOf inbuf

Invoke ExitProcess,0

End Start

**Пример 3.** Написать программу поиска в исходной строке символа, который указан в константе.

; Template for console application

.586

.MODEL flat, stdcall

OPTION CASEMAP:NONE

Include kernel32.inc

Include masm32.inc

IncludeLib kernel32.lib

IncludeLib masm32.lib

.CONST

MsgExit DB 0AH,0DH,"Press Enter to Exit",0AH,0DH,0

**IsxStroka db 'Primer of string mov'** ;строка-источник

**Symbol db 's' ; символ для поиска**

yes db 'yes' ; пересылается в строку для вывода результата если символ обнаружен

no db 'not' ; Пересылается в строку для вывода если символ не обнаружен

.DATA

.DATA?

inbuf DB 100 DUP (?)

Result db 22 DUP (' ') ;строка-результат

.CODE

Start:

**cld** ;сброс флага DF - обработка строки от начала к концу

**lea edi,IsxStroka** ;загрузка в di смещения строки-источника

**mov ecx,20** ; длина строки источника

**mov al,Symbol** ; загрузка символа для поиска

**repne scasb**

**je equal** ; Нашли символ

mov ecx,3 ; Не нашли символ

lea esi,no

lea edi,Result ; подготовка вывода сообщения Нет символ не найден

rep movsb

jmp kon

**equal:** mov ecx,3

lea esi,yes

lea edi,Result ; подготовка вывода сообщения Да символ найден

rep movsb

kon:

XOR EAX,EAX

**Invoke StdOut,ADDR Result** ; вывод результата поиска

Invoke StdOut,ADDR MsgExit

Invoke StdIn,ADDR inbuf,LengthOf inbuf

Invoke ExitProcess,0

End Start

**Пример 5.** Дана строка, состоящая из слов, разделенных пробелом, после последнего слова – пробел. Написать программу определения количества слов, длина которых больше трех символов. Строку ввести с клавиатуры, результат вывести на экран.

; Template for console application

.586

.MODEL flat, stdcall

OPTION CASEMAP:NONE

Include kernel32.inc

Include masm32.inc

IncludeLib kernel32.lib

IncludeLib masm32.lib

.CONST

MsgExit DB 13,10,"Press Enter to Exit",0AH,0DH,0

Msg2 DB 13,10,"input string slova i probely ",0AH,0DH,0

.DATA

strv db 50 dup(?) ; буфер для ввода строки

dlina dw 3

rez dd ? ; байты для вывода результата

string db 13,10,"kol. slov >3 simv.= " ; комментарий при выводе

rezstr db 16 dup(?) ; буфер для вывода результата

.DATA?

inbuf DB 100 DUP (?)

.CODE

Start:

mov ecx,0

Invoke StdOut,ADDR Msg2 ; Вывод подсказки

**Invoke StdIn, ADDR strv,LengthOf strv ; вводим строку**

**Invoke StripLF,ADDR strv ; заменяем #13#10 нулем**

**;** *Подсчет длины введенной строки до 0*

lea edi,strv ; загружаем адрес строки в edi

mov Ecx,50 ; загружаем размер буфера ввода для подсчета длины строки

mov al,0 ; загружаем 0 для дальнейшего поиска

repne scasb ; ищем завершающий 0 в строке

mov ax,50

sub ax,cx ; вычитаем из размера буфера остаток в cx

mov cx,ax ; полученная разница – длина строки +1

dec cx ; корректируем cx для получения истинной длины строки

; *Подсчет слов с указанной длиной (> 3 символов)*

**lea edi,strv** ; загружаем адрес строки в edi

**mov al,’ ‘** ; загружаем в al пробел для поиска

**mov ebx,0** ; обнуляем счетчик слов >3

**cld**

**cic1: cmp cx,0** ; цикл с предусловием конец строки достигнут?

**je con**  ; если да, переход на вывод результатов

**mov dx,0;** обнуляем счетчик длины слова

**cic2: scasb** ; проверяем очередной символ на пробел

**je consl** ; если пробел – переходим на проверку соответствия длины слова требуемой

**inc dx** ; если нет – увеличиваем длину строки

**dec cx** ; уменьшаем счетчик повторов

**jmp cic2** ; переход на следующую итерацию цикла поиска пробела

**consl: cmp dx,dlina** ; если конец слова сравниваем длину слова с заданной

**jle prod** ; если меньше или равна переходим к следующему слову

**inc bx** ; иначе увеличиваем счетчик слов

**prod: dec cx** ; уменьшаем счетчик повторов и переходим к новому слову

**jmp cic1**

con: mov rez,ebx ; полученное количество слов записываем в поле результата

XOR EAX,EAX

Invoke dwtoa,rez,ADDR rezstr ; преобразуем число в строку

Invoke StdOut,ADDR string ; выводим результат работы

Invoke StdOut,ADDR MsgExit

Invoke StdIn,ADDR inbuf,LengthOf inbuf

Invoke ExitProcess,0

End Start