Лабораторная работа №7

Цепочечные команды

Цель работы: изучение цепочечных команд для работы с одномерными массивами и строками

Цепочечные команды предназначены для обработки одномерных массивов и позволяют выполнять их сравнение, копирование, инициализацию, а также поиск элементов. Элементы массива могут иметь размер байта, слова или двойного слова. Если команда обрабатывает один массив, то начальный адрес массива необходимо поместить в один из индексных регистров **ESI** или **EDI**; если два массива, то адрес первого (источника) – в **ESI**, второго (приёмника) – в **EDI**. Механизм работы данных команд следующий. Выполняется последовательная обработка элементов массивов, начиная с адресов **ESI** и **EDI** до тех пор, пока не будет выполнено условие для остановки работы команды, например, элемент будет найден (для поиска), или выполнено нужное число итераций. Условие остановки определяется *префиксом повторения*. Существуют следующие виды префиксов:

REP – повторять, пока **ECX** ≠ 0.

REPE/**REPZ** – повторять пока **ECX** ≠ 0 и **ZF** = 1 (элементы равны).

REPNE/**REPNZ** – повторять пока **ECX** ≠ 0 и **ZF** = 0 (элементы не равны).

Префикс **REP** используется, если цепочечная команда обрабатывает количество элементов, заданное в регистре **ECX**. При переходе к каждому следующему элементу значение в **ECX** уменьшается на единицу. При использовании **REPNE**/**REPNZ** выполнение команды может завершиться досрочно, если обрабатываемые элементы на какой-то итерации окажутся равны и **ZF** будет установлен. **REPE**/**REPZ** завершает работу, если элементы не равны или **ECX** = 0.

MOVS копирует элемент массива с адресом **ESI** в ячейку памяти с адресом **EDI**. Команда **MOVS** имеет два операнда, но фактически они нужны только для определения размерности операнда. Транслятор определяет размерность и преобразует её в одну из трёх команд: **MOVSB** (пересылка байта), **MOVSW** (пересылка слова) и **MOVSD** (пересылка двойного слова). Команду **MOVS** хорошо использовать для копирования больших блоков памяти, размер которых заранее известен. В следующем примере осуществляется копирование массива двухбайтовых целых чисел длиной 10.

```
…
.data
   array_source dw 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
   array_dest dw 10 dup(?)
 .code
 start:
  CLD
  MOV ESI, offset array_source   ; В ESI - адрес источника
  MOV EDI, offset array_dest     ; В EDI - адрес приёмника
  MOV ECX, 10         ; Количество пересылок
  REP MOVSW           ; Копирование массива слов
  …
 end start
```

Таким образом, цепочечные команды неявно работают с регистрами **ESI**, **EDI**, **ECX**. Они могут применяться с префиксом повторения и без. В случае отсутствия префикса

команда выполняется один раз. Цепочечная команда на каждом повторении увеличивает или уменьшает значение в **ESI** и/или **EDI** на размер элемента массива. Если команда используется с префиксом, то **ECX** уменьшается на единицу. В вышеприведённом примере базовый элемент массива занимает 2 байта, соответственно индексные регистры увеличивали значение на каждом повторении на 2, а после всех повторений значения изменились на 20. Если флаг **DF** (флаг направления) установлен в 0, то цепочечные команды будут увеличивать значения в **ESI** и/или **EDI** после каждого повторения или уменьшать, если **DF** = 1. Для установки флага **DF** используется команда **STD**, для сброса – **CLD**.

Команды **CMPS**/**CMPSB**/**CMPSW**/**CMPSD** сравнивают элементы в памяти с адресами **ESI** и **EDI**. Механизм работы данных команд подобен **CMP**. Они также производят вычитание элементов, но при этом изменяют только флаги. Поэтому после выполнения **CMPS** результат последнего сравнения можно анализировать с помощью команд условного перехода. Команду **CMPS** можно использовать со всеми тремя префиксами:

- **REP CMPS** сравнивает элементы, пока **ECX** ≠ 0.
- **REPE**/**REPZ CMPS** сравнивает элементы, пока они равны (**ZF**=1), и **ECX** ≠ 0.
- **REPNE**/**REPNZ CMPS** наоборот, сравнивает элементы, пока они различны (**ZF**=0), и **ECX** ≠ 0.

В следующем примере сравниваются 2 блока памяти размером 5 байт.

```
…
.data
  array_1 db 1, 2, 5, 4, 5
  array_2 db 1, 2, 4, 5, 6
.code

start:
 MOV ESI, offset array_1
 MOV EDI, offset array_2
 MOV ECX, 5 ; Количество элементов в массивах
 REPE CMPSB ; Сравнение массивов
 ; Анализ результата сравнения
 JA j1; array_1 > array_2
 JB j2; array_1 < array_2
 JE j3; Массивы равны
 …
```

Цепочечные команды сравнения удобно использовать для сравнения блоков памяти известной длины. При этом сравнение одного и того же объёма памяти командой **CMPSW** производится быстрее, чем **CMPSB**. Самым быстрым способом является сравнение блоков командой **CMPSD**.

Команды **SCAS**/**SCASB**/**SCASW**/**SCASD** сравнивают байт/слово/двойное слово в памяти по адресу **EDI** с элементом, помещённым в **AL**/**AX**/**EAX**. Если используется префикс, то команды перебирают все элементы массива до тех пор, пока не будет найден элемент, равный **AL**/**AX**/**EAX** или отличный от него. Адрес начала массива должен быть помещён в **EDI**. Возможны следующие варианты использования команды **SCAS** с префиксами:

- **REPE**/**REPZ SCAS**, как и команда сравнения, перебирает элементы массива до тех пор, пока эти элементы равны содержимому **AL**/**AX**/**EAX** и **ECX** ≠ 0.
- **REPNE**/**REPNZ SCAS**, перебирает элементы массива, пока они не равны содержимому аккумулятора и **ECX** ≠ 0. По сути, при использовании префикса **REPNE**/**REPNZ** с командой **SCAS** происходит поиск первого элемента **AL**/**AX**/**EAX** в массиве.

Команды **STOS**/**STOSB**/**STOSW**/**STOSD** помещают в ячейку памяти с адресом **EDI** содержимое **AL**/**AX**/**EAX**. При использовании префикса **REP** весь массив, начиная с адреса **EDI**, заполняется одинаковыми значениями, равными содержимому аккумулятора.

Данную команду удобно использовать для инициализации блоков памяти. Рассмотрим пример инициализации большого блока памяти одинаковыми числами FFh.

```
…
.data
  memory_block db 1024 dup(?)
.code
 start:
 XOR EAX, EAX ; Обнулить EAX
 NOT EAX ; Инвертировать содержимое EAX
 MOV ECX, 1024/4 ; Количество повторений
 MOV EDI, offset memory_block
 REP STOSD

 …
```

Команды **LODS**/**LODSB**/**LODSW**/**LODSD** пересылают байт/слово/двойное слово из области памяти с адресом, содержащимся в **ESI**, в регистр **AL**/**AX**/**EAX** и изменяют **ESI**. Префикс в данной команде используется редко, но саму команду удобно применять после команды **CMPS** для загрузки последнего сравниваемого числа из массива в аккумулятор.

Для работы с портами ввода/вывода также предусмотрены цепочечные команды. **INS**/**INSB**/**INSW**/**INSD** вводят элемент из порта. Номер порта должен быть размещён в регистре **DX**, адрес массива, в который вводятся элементы – в **EDI**, количество вводимых элементов – в **ECX**.

Вывод массива с начальным адресом в **ESI** в порт реализуют команды **OUTS**/**OUTSB**/**OUTSW**/**OUTSD**. Длина массива должна быть помещена в **ECX**, номер порта – в **DX**.

Так как цепочечные команды модифицируют регистр **ECX** на каждом повторении, уменьшая его содержимое на единицу, очень удобно для организации циклов в программах, включающих цепочечные команды использовать команду условного перехода **JECXZ**. **JECXZ <метка>** передаёт управление по метке, если **ECX** = 0.

Задания для выполнения к работе

Написать программу для решения соответствующего варианта задания. По умолчанию, если в задании не оговорено, считать что в тексте могут быть только буквы русского алфавита, латинские буквы, цифры, пробелы и нулевой символ как признак окончания строки. Пробелов между словами может быть несколько. Необходимые операции копирования, сравнения, поиска и другие организовать в виде подпрограмм, используя цепочечные команды. Если в результате преобразования длина строки увеличивается, то изначально зарезервировать для неё большой объём памяти.

Пример выполнения задания:

Подсчитать количество слов вводимой с клавиатуры строки.

```
.486
 .model flat, stdcall
 option casemap: none

 include d:\masm32\include\kernel32.inc
 include d:\masm32\include\msvcrt.inc
 includelib      d:\masm32\lib\kernel32.lib
 includelib      d:\masm32\lib\msvcrt.lib

 .data
```

```
    str1 db 1024 dup(?)
    format_str db "%d", 0
  .code
; Функция для подсчёта количества слов в строке, разделённых пробелами
get_word_count proc
  ; Сохранить значения используемых регистров в стеке
  PUSH EBX
  PUSH ECX
  PUSH EDI
  ; Поместить в EDI адрес обрабатываемой строки
  MOV EDI, [ESP+16]
  ; Сначала нужно найти длину строки, поэтому нужно загрузить в AL искомый символ. Признак окончания строки - нулевой символ
  MOV AL, 0
  ; Поместить в ECX -1, чтобы инициализировать его максимальным значением, т.к. число итераций не известно, а условием остановки цепочечной команды является нахождение первого нулевого байта
  MOV ECX, -1
  ; Выполнить поиск символа, помещённого в AL
  REPNE SCASB
  ; Теперь нужно вычислить длину строки
  MOV ECX, EDI
  ; Адрес нулевого байта известен. Он находится в EDI. Адрес начала строки - в стеке
  ; Разница этих адресов и есть длина строки
  SUB ECX, [ESP+16]
  ; EBX - счётчик количества слов
  XOR EBX, EBX
  ; Поместить в EDI адрес начала строки
  MOV EDI, [ESP+16]
  ; Поиск пробела
  MOV AL, ' '
  ; Пропустить все пробелы в начале строки, т.е. выполнять команду, пока в ячейках памяти по адресу EDI пробелы
  REPE SCASB
 j1:
  ; Если ECX = 0, то конец алгоритма
  JECXZ j_end
  ; Условие остановки команды - ECX = 0 или найден пробел (разделитель слов)
  REPNE SCASB
  ; Если между словами несколько пробелов, то нужно их все пропустить
  REPE SCASB
  ; Увеличить количество слов на единицу
  INC EBX
  ; Переход в начало цикла к следующему слову
  JMP j1
 j_end:
  ; Поместить результат - количество слов в EAX
  MOV EAX, EBX
  ; Восстановление регистров из стека
  POP EDI
  POP ECX
  POP EBX
  ; Возврат из подпрограммы
  RET 4
get_word_count endp

 start:
  ; Вызов стандартной функции crt_gets для ввода строки с клавиатуры
  PUSH offset str1
  CALL crt_gets
  ADD ESP, 4 ; Очистка стека от аргумента
```

```
; Вызов функции для подсчёта количества слов в строке
PUSH offset str1
CALL get_word_count

; Вывод результата на экран
PUSH EAX
PUSH offset format_str
call crt_printf
ADD ESP, 8

call crt__getch ; Задержка ввода с клавиатуры
push 0
call ExitProcess ; Выход из программы
end start
```

Вариант	Задание
1	Вывести слова исходного текста в лексикографическом порядке. Считать, что слова имеют одинаковую длину.
2	Удалить из текста повторяющиеся друг за другом слова.
3	Если в тексте встречаются повторяющиеся слова, то вывести эти слова и их номера.
4	Преобразовать строку, удалив из неё повторяющиеся пробелы.
5	Вывести все пары слов текста, которые отличаются только одной буквой.
6	Преобразовать текст, внеся в него следующие исправления: удалить повторяющиеся запятые, оставить после каждой запятой один пробел, перед первым словом и после последнего всё удалить.
7	С клавиатуры вводятся слово *a* и строка. Исправить в строке одиночные ошибки в словах *a*. Считать, что длины исходного слова и исправляемого равны.
8	Проверить, упорядочены ли слова данной строки по не возрастанию/не убыванию.
9	Преобразовать строку, удалив из неё слова с одинаковыми символами.
10	Преобразовать строку, заключив самое длинное слова в квадратные скобки.
11	Преобразовать строку, заменив второе и последующие повторяющиеся слова символами подчеркивания «_».
12	Преобразовать строку, заменив два или более идущих по порядку пробелов знаком табуляции. Код символа табуляции – 9.
13	Преобразовать строку, внеся в него следующие исправления: если слово состоит из двух одинаковых половинок, то заключить его в угловые скобки.
14	Преобразовать строку, заменив в ней слово «один» цифрой «1», «два» – цифрой «2» и т.д. до «девять» – цифрой «9».
15	Найти самое часто встречающееся слово данной строки.
16	Дано число *n* и строка. Заменить в строке все буквы слов, имеющие длину *n* символами подчёркивания «_».
17	Удалить из строки слова с длиной *n*.
18	Вывести множество слов данного текста.
19	Преобразовать строку, поставив после каждой запятой пробел, если его нет.
20	Преобразовать строку, поставив между словами «столько же» и «сколько» запятую, если её нет.
21	Преобразовать строку, поставив после каждой точки пробел, если его нет. Если предложение начинается со строчной буквы, то заменить её прописной.
22	Вывести все пары слов данной строки, которые отличаются только одним символом и имеют одинаковую длину.

23	Преобразовать строку, удалив из неё первое самое длинное слово.
24	Преобразовать строку, заключив самые короткие слова в фигурные скобки.
25	С клавиатуры вводятся слова *a* и *b*, а также строка. Преобразовать строку, заменив в ней все слова *a* словами *b*.