ПРАКТИЧЕСКАЯ РАБОТА № 9.

**Тема: *«РАЗРАБОТКА ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ КОМАНД СТРОКОВЫХ ПРИМИТИВОВ».***

**Цель:**

* изучение особенностей выполнения команд обработки строк;
* оценка преимущества цепочечных команд над командами, работающими с отдельными элементами.

**ХОД РАБОТЫ:**

1. Повторить приведенный ниже теоретический материал.
2. Разобрать примеры приведенных программ, реализующих обработку строк.
3. Внимательно изучить задание своего варианта на обработку цепочек элементов.
4. Разработать блок-схему программы, обрабатывающей цепочки элементов в памяти.
5. Задать цепочку элементов (строку) в формате DB и реализовать ее обработку в соответствии с заданием. Длина строки N. Исходные данные задать самостоятельно (либо получить у преподавателя).
6. Произвести тестовые проверки, отметить нормальные и аномальные результаты, сделать анализ результатов.
7. Сравнить результаты, полученные в результате работы программы, с рассчитанными вручную.
8. Ответить на контрольные вопросы.
9. Оформить отчет по практической работе, включающий тему и цель работы, листинги программ, тестовые примеры, ответы на контрольные вопросы.

**ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ.**

**Цепочки элементов в памяти. Команды, работающие с цепочками.**

***Цепочкой* памяти называется последовательность элементов размером в слово или двойное слово.** Если размер отдельного элемента – байт, обычно последовательность таких элементов называют не цепочкой, а ***строкой символов***. Команды, работающие с цепочками (строками), обрабатывают сразу блок памяти, причем содержимое этих блоков для CPU не имеет никакого значения. Это могут быть символы, числа и т.п. Главное, чтобы размерность элементов была 8 бит, 16 бит или 32 бита.

Различные языки высокого уровня используют разные форматы хранения строк символов. Рассмотрим несколько основных форматов, используемых наиболее часто:

**Дескриптор длины.** Байт дескриптора длины может храниться в начале строки. В Turbo Pascal, например, этот формат позволяет изменять длину строки от 0 до числа байтов, указанного при объявлении строки. Неиспользуемые позиции в строке являются неопределенными и отмечены знаками вопроса:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| \* | S | T | R | I | N | G |  | T | W | O | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |

|

0Ah

На ассемблере такая строка может быть объявлена следующим образом:

db 0Ah, "STRING TWO", 10 dup (?)

**Строки с нулевым окончанием.** Строки **ASCIIZ**, или строки с нулевым окончанием, представляют собой последовательность символов, заканчивающихся нулевым байтом. Действительная длина строки может находиться в диапазоне от нуля до числа байтов, установленного при определении строки. Неиспользуемые позиции считаются неопределенными и отмечены знаками вопроса. Такой формат используется в языке С:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | T | R | I | N | G |  | T | H | R | E | E | \* | ? | ? | ? | ? | ? | ? | ? |

|

нулевой байт

На ассемблере такая строка может быть объявлена следующим образом:

db "STRING THREE", 0, 7 dup (?)

Программа на языке ассемблера должна быть совместима со всеми форматами.

**Общие положения по обработке строк в ассемблере.**

1. В мнемонике команд обработки строк всегда содержится буква **S** (String – строка). Она является последней или предпоследней буквой.
2. Содержимое строки для микропроцессора не имеет никакого значения. Основное, что имеет значение – это длина операнда.
3. Строка в **базовом ассемблере** может обрабатываться побайтно (последняя буква в команде будет B) или пословно (W).
4. Строка может обрабатываться группой (цепочкой), тогда перед командой появляется префикс **REPx** (REPeat). Количество повторений должно находиться в регистре CX. Этот префикс алгоритмически подобен команде LOOPx.
5. Строка-приемник должна находиться обязательно в дополнительном сегменте памяти ES со смещением DI (адресация ES:DI).
6. Строка источник должна находиться в сегменте данных DS со смещением SI (адресация DS:SI).
7. В процессе циклического выполнения команд указатели SI и DI **автоматически модифицируются** в зависимости от длины элемента строки и значения флага DF. Если **DF=0**, значения SI и DI инкрементируются на длину элемента (строка обрабатывается слева направо – в сторону старших адресов). Если **DF=1**, значения SI и DI декрементируются на длину элемента (строка обрабатывается справа налево – в сторону младших адресов).
8. Флаг направления **DF** очищается или устанавливается, соответственно, командами **CLD** или **STD.**
9. Длина строки в базовом ассемблере <= 64Кбайт.
10. Команды сравнения или сканирования строк устанавливают значения флагов аналогично команде CMP.
11. Команды обработки строк **могут иметь два операнда памяти**, что не допускается в других командах.

В системе команд CPU рассмотрим 5 основных операций-примитивов по обработке цепочек. Каждая из них реализуется в CPU тремя командами соответственно для элементов размером в байт, слово или двойное слово. **Особенностью всех цепочечных команд является автоматическое перемещение к следующему элементу цепочки за счет автоматического инкремента (декремента) текущего адреса на длину элемента цепочки**. Использование команд строковых примитивов дает большое повышение производительности процессора (особенно i8086/8088).

**Таблица. *Команды строковых примитивов.***

|  |  |  |
| --- | --- | --- |
| **Команда** | **Описание** | **Операнды** |
| **MOVS**   * MOVSB * MOVSW * MOWSD | Перемещает строки данных. Копирует байт, слово, двойное слово из одного участка памяти в другой.  Перемещает (копирует) байт с шагом инкрем.1 для SI и DI  Перемещает (копирует) слово с шагом инкр. 2 для SI и DI  Перемещает (копирует) двойное слово с шагом инкремента 4 для SI и DI (только для CPU i386+) | MOVS [DI], [SI] |
| **CMPS**   * CMPSB * CMPSW * CMPSD | Сравнивает строки данных. Сравнивает в памяти байты, слова, двойные слова.  Сравнивает байты с шагом инкремента 1 для SI и DI  Сравнивает слова с шагом инкремента 2 для SI и DI  Сравнивает двойные слова с шагом инкремента 4 для SI и DI (только для CPU i386+) | СMPS [SI], [DI] |
| **SCAS**   * SCASB * SCASW * SCASD | Сканирует строки данных. Сравнивает регистры AL, AX или EAX с содержимым памяти, изменяя регистр флагов.  Сканирует байт из AL с шагом инкремента 1 для DI  Сканирует слово из AX с шагом инкремента 2 для DI  Сканирует двойное слово из EAX с шагом инкремента 4 для DI (только для CPU i386+) | SCAS [DI], AL  SCAS [DI], AX  SCAS [DI], EAX |
| **STOS**   * STOSB * STOSW * STOSD | Сохраняет строку данных: сохраняет содержимое регистров AL, AX или EAX в памяти.  Сохраняет байт из AL с шагом инкремента 1 для DI  Сохраняет слово из AX с шагом инкремента 2 для DI  Сохраняет двойное слово из EAX с шагом инкремента 4 для DI (только для CPU i386+) | STOS [DI], AL  STOS [DI], AX  STOS [DI], EAX |
| **LODS**   * LODSB * LODSW * LODSD | Загружает аккумулятор из строки. Загружает байт, слово или двойное слово в AL, AX или EAX из памяти.  Загружает байт в AL с шагом инкремента 1 для SI  Загружает слово в AX с шагом инкремента 2 для SI  Загружает двойное слово в EAX с шагом инкремента 4 для SI (только для CPU i386+) | LODS AL, [SI]  LODS AX, [SI]  LODS EAX, [SI] |

В регистре ES необходимо указать начало сегмента данных еще до того, как будут использоваться команды работы со строками. Регистры ES и DS могут быть инициализированы одновременно. Например:

***mov ax, @data ; установить адрес сегмента данных***

***mov ds, ax ; инициализировать DS***

***mov es, ax ; инициализировать ES.***

Каждая команда строковых примитивов имеет три допустимых формата. Общий формат может использоваться с байтами, словами или двойными словами. Команды, оканчивающиеся буквой **B** (MOVSB, CMPSB…), используют только 8-разрядные операнды. Команды, оканчивающиеся на букву **W** (MOVSW, CMPSW…), используют только 16-разрядные операнды и т.п. Эти суффиксы определяют шаг инкрементации индексных регистров. **Общий формат** (без суффиксов) не привязан к размеру операндов, (например, MOVS, CMPS), поэтому операнды должны быть явно определены.

Если определен размер отдельного операнда, нет необходимости указывать этот операнд в команде. Но в любом случае в регистрах SI и DI **должно быть** **установлено смещение** операндов.

Например,

***mov si, offset source ; SI указывает на отправителя***

***mov di, offset dest ; DI указывает на получателя***

***movsb ; копирует отправитель в получатель***

Итак, при работе с цепочками возникает необходимость **получения полного указателя данных в памяти**, т.е. адреса сегмента и значения смещения внутри сегмента. В этом могут помочь следующие команды:

**lea *назначение, источник* –** загрузка эффективного адреса

**lds *назначение, источник* –** загрузка указателя в регистр сегмента данных **ds**

**les *назначение, источник* –** загрузка указателя в регистр дополнит. сегмента данных **es**

Например:

***.data***

***…***

***str1 db "Первая строка"***

***str2 db "Вторая строка"***

***adress1 dd str1***

***…***

***.code***

***…***

***lea si, str1***

***lea di, str2***

***les bx, adress1 ; полный указатель на str1 в пару ES:BX***

( Когда используются явные операнды, в команде необходимо указывать регистр сегмента (ES:), чтобы обеспечить доступ к операндам по их смещениям. Например,

**movs es:dest, source ; копирование из отправителя в получатель** )

**Использование префикса повторения**. Команды строковых примитивов перемещают только один байт или слово, но такое положение можно изменить с помощью префикса. Такой префикс заставляет команду выполняться несколько раз, делая возможным перемещение строк. Для этого используются такие префиксы повторения:

**REP - повторять, пока CX > 0**

**REPZ, REPE - повторять, пока флаг нуля установлен (ZF=1) и СХ > 0**

**REPNZ, REPNE - повторять, пока флаг нуля сброшен (ZF<>0) и СХ>0**

При использовании префиксов необходимо поместить в регистр СХ (счетчик) нужное значение. Например:

***mov si, offset string1 ; SI указывает на отправителя***

***mov di, offset string2 ; DI указывает на получателя***

***mov cx, 10 ; устанавливает счетчик в 10***

***rep movsb ; перемещает 10 байт***

Префикс повторения транслируется ассемблером в один байт машинного кода, который предваряет команду строкового примитива. Показанные выше команды в машинных кодах будут выглядеть так:

***BE 0000 R mov si, offset string1***

***BF 000A R mov di, offset string2***

***B9 000A mov cx, 10***

***F3/ A4 rep movsb***

Символ **R** поставлен потому, что пока не известны смещения операндов, т.к. **сегмент данных не определен до момента выполнения**. Префикс **REP** преобразовывается в машинную команду **F3**, а из команды **MOVSB** получается только один байт **A4**.

**Флаг направления.** Команды строковых примитивов используют флаг направления, чтобы определить регистры SI и DI будут инкрементироваться или декрементироваться при выполнении команды. Флаг направления изменяется командами:

**CLD** ; сбросить флаг направления, направление – вверх

**STD** ; установить флаг направления, направление – вниз.

**Пример копирования байтов**: 10 байт будут скопированы из поля отправителя в поле получателя. После копирования оба поля будут содержать одинаковые строки, а регистры SI и DI будут указывать на первый байт после конца строки.

***…***

***.data***

***source db 'ABCDEFGHIJ'***

***dest db 10 dup (?)***

***….***

***.code***

***….***

***mov ax, @data***

***mov ds, ax***

***mov e s, ax***

***cld ; направление вверх***

***mov cx, 10 ; установить счетчик в 10***

***mov si, offset source ; DS:SI указывает на отправителя***

***mov di, offser dest ; ES:DI указывает на получателя***

***rep movsb***

***…***

(Наличие префикса **rep** приводит к проверке регистра СХ до выполнения команды **movsb**, поэтому, если СХ=0, команда не будет выполняться.)

В программах, работающих в системах с CPU i8088/i8086, благодаря командам строковых примитивов значительно повышается производительность по сравнению с командой **mov** при выполнении аналогичных задач. Это объясняется несколькими причинами:

* Машинные команды строковых примитивов занимают всего один байт, т.е. значительно меньше места в предварительной очереди команд, чем соответствующие команды mov.
* Команды строковых примитивов автоматически инкрементируют индексные регистры. Поэтому нет необходимости организовывать отдельный счетчик для подсчета последовательности байтов, слов или двойных слов.
* С помощью префикса **rep** команды строковых примитивов могут быть повторены 65535 раз, и при этом не используются команды перехода, на которые обычно тратится много времени. Команды перехода в процессорах 80888 вызывают очистку предварительной очереди команд и повторную ее загрузку.

***Например***, необходимо скопировать все значения из массива **source** в массив **dest**:

**; с использованием команды mov**

**.data**

**ARRAY\_SIZE = 1000**

**souce dw ARRAY\_SIZE dup (?)**

**dest dw ARRAY\_SIZE dup (?)**

**.code**

**…**

**mov ax,@data**

**mov ds,ax**

**mov si,0**

**mov cx, ARRAY\_SIZE**

**L1: mov ax, souce [si]**

**mov dest [si], ax**

**add si,2**

**loop L1**

***; с использованием команды movsw***

***mov ax,@data***

***mov ds,ax***

***move es,ax***

***mov si, offset source***

***mov di, offset dest***

***mov cx, ARRAY\_SIZE ; счетчик повторений***

***cld ; направление вверх***

***rep movsw***

**Отметим еще раз ОСОБЕННОСТИ работы с перечисленными в таблице командами.**

Команды перемещения (копирования) **MOVS (Move String Data)** переносят данные из места отправления, адрес в DS:SI, в место получения, адрес в ES:DI.

Команда **CMPS** сравнивает операнд-отправитель (DS:SI) с операндом-получателем (ES:DI). Если строка операнда-отправителя меньше, чем операнда-получателя, CF=1. Если строки равны, ZF=1. Если строка операнда-отправителя > операнда-получателя, CF=0 и ZF=0. Условные переходы, используемые командой CMPS, приведены в таблице:

|  |  |  |
| --- | --- | --- |
| **Условие** | **Без знака** | **Со знаком** |
| source < dest | JB | JL |
| source < = dest | JBE | JLE |
| source > < dest | JNE (JNZ) | JNE (JNZ) |
| source = dest | JE (JZ) | JE (JZ) |
| source > = dest | JAE | JGE |
| source > dest | JA | JG |

Следует быть особенно внимательными при использовании команды CMPS с условными переходами! Порядок операндов отличается от команды CMP, где сравниваются операнд-получатель с операндом-отправителем (вычитание источника из приемника). В команде CMPS, наоборот, используется **вычитание операнда-получателя (приемника) из операнда-отправителя (источника).** Например:

**mov ax, 10**

**cmp ax,5 ;** высчитывает AX - 5

**cmps es:dest, source ; высчитывает source – dest !!!**

**Cравнение чисел со знаком.** Команда **CMPS (Compare String)** может сравнивать два списка целых чисел со знаком. В следующем примере каждое число в массиве ***array1***  сравнивается с соответствующими числами в массиве ***array2***.

**.data**

**array1 dw 3, 9, -7, 15, 10**

**array2 dw 3, 9, 8, 15, 10**

**.code**

**cld**

**mov si, offset array1**

**mov di, offset array2**

**mov cx, 5**

**repe cmpsw**

**jl array1\_smaller**

Команда **SCAS (Scan String)** сравнивает значения регистров AL/AX/EAX с байтом, словом или двойным словом по адресу в ES:DI (с операндом-получателем). Эти команды особенно удобны при работе с отдельными символами в длинной строке. В комбинации с префиксом **repe** (или **rep**) можно производить сканирование строки, только когда значение в регистре-аккумуляторе совпадает со значением в памяти. С префиксом **repne** производится сканирование до тех пор, пока значение в регистре-аккумуляторе не совпадает со значением в памяти или СХ=0.

**Сканирование для совпадающих символов.** В следующем примере строка ***alpha*** сканируется до тех пор, пока символ F не будет найден. Когда символ F будет обнаружен, регистр DI будет указывать на первый байт после этого символа и **значение регистра надо декрементировать, чтобы он указывал на символ F**.

***.data***

***alpha db 'ABCDEFGH',0***

***.code***

***mov di, seg alpha***

***mov ds, di***

***mov di, offset alpha ; сейчас ES:DI указывает на строку***

***mov al, 'F' ; поиск символа 'F'***

***mov cx, 8 ; установили счетчик***

***cld ; направление вверх***

***repne scasb ; повторять пока не равно***

***; выход, если ZF=1 (т.е. символ найден и получили равенство АХ***

***; и приемника) или CX=0***

***jnz exit***

***; если выход по ZF=1 (нашли символ), уходим на exit***

***; если выход по CX=0 (символ не найден), идем на следующую команду***

***dec di ; найден – вернуться на DI***

Вы обратили внимание, что сразу после выполнения команды SCASх еще невозможно определить, что именно стало причиной ее завершения: то, что CX=0, или то, уже найден соответствующий символ, причем и то и другое может произойти одновременно. При использовании префикса **repnz (repne)** обычно используют команду **jnz** для перехода, когда символ найден. В этом случае руководствуются следующим принципом **– *если флаг нуля сброшен, символ не найден, а установленный флаг нуля говорит о том, что символ найден.***

Команды сохранения **STOS (Store in String)** сохраняют содержимое регистров AL/AX/EAX в памяти по адресу ES:DI. Также можно использовать STOS для инициализации памяти единственным значением.

***Например:*** инициализировать каждый байт строки str1 значением 0Ah.

***.data***

***str1 db 100 dup (?)***

***.code***

***mov di, seg str1***

***move es, di ; определим адрес сегмента***

***mov al, 0ah ; заносим значение для заполнения байтов строки***

***mov di, offset str1 ; определим адрес получателя ES:DI***

***mov cx, 100 ; счетчик символов***

***cld ; направление вверх***

***rep stosb ; заполнить содержимым***

Команды загрузки **LODS (Load String)** загружают из памяти по адресу, указанному в DS:SI, байт или слово **в** регистры AL/AX/EAX. Префиксы повторения редко используются с командами LODS, потому что каждое новое значение, загружаемое в регистры AL/AX, разрушает их прежние значения. Вместо отдельной команды LODSB можно использовать следующие две команды:

**mov al, [si]**

**inc si**

**Следует помнить, что строковые команды имеют следующие особенности**:

- позволяют обрабатывать два операнда, находящиеся в оперативной памяти. Оба операнда заданы по умолчанию;

- могут работать в паре со специальными командами, которые называются префиксами повторения. В паре с префиксами повторения строковые команды работают в цикле и обрабатывают поля (цепочки) байт или слов. Поэтому эти команды называют также командами обработки цепочек;

- при выполнении команды анализируется флаг направления DF. Этот флаг не является признаком результата выполнения каких-либо команд. Для работы с ним предусмотрены специальные команды cld - сброс флага DF, std - установка флага DF.

- в командах неявно заданы операнды: первый называется приемником и адресуется регистрами ES:DI, второй называется источником и адресуется регистрами DS:SI. Например, команда movsb(w) выполняет передачу байта (слова) источника в поле приемника. Если DF=0, то передача в прямом направлении, иначе (DF=1) - в обратном направлении.

**ПРАКТИЧЕСКОЕ ЗАДАНИЕ**

***ЗАДАНИЕ 1.***

**Вариант 1.** строку символов. Замените в ней все 'A' на 'AAA'. Результирующую строку сохранить в памяти.

**Вариант 2.** Инициализируйте массив ASCII – кодов. Замените входящие в массив заглавные латинские буквы на строчные.

**Вариант 3.** Напишите программу, которая заполняет строку с нулевым окончанием последовательностью случайных символов в диапазоне от "А" до "Z". Замените центральный символ строки первым. Конечный и промежуточный результаты работы сохранить в памяти.

**Вариант 4.** Заполните два массива ASCII – кодов одинаковой длины. Сформируйте третий массив, состоящий из попарно совпадающих элементов заданных массивов. Результаты работы сохранить в памяти.

**Вариант 5.** Инициализируйте строку, состоящую из одного слово, проверить будет ли оно читаться одинаково справа налево и слева направо (т.е. является ли оно палиндромом). Ответ сохранить в памяти.

**Вариант 6.** Инициализируйте строку символов, содержащую несколько точек. Выделить подстроку между первой и второй точкой. Выделенную строку сохранить в памяти.

**Вариант 7.** Инициализируйте строку символов, длиной **до** 20 байт. Ввести слово, размером **до** 10 байт. Содержит ли данная строка символов это слово? Результаты работы сохранить в памяти.

**Вариант 8.** Инициализируйте строку символов, состоящую из слов, разделенных пробелом. Подсчитайте количество слов. Количество слов сохранить в памяти.

**Вариант 9.** Инициализируйте массив ASCII – кодов. Сформируйте по его содержимому массив ASCII –кодов в обратной последовательности. Подсчитайте количество символов "А", "В" и "С" в заданном массиве кодов. Результаты работы сохранить в памяти.

**Вариант 10.** Инициализируйте строку символов. Группы символов в ней, разделенные пробелом, считаются словами. Определить сколько слов содержат хотя бы одну букву "В"? Количество слов с буквой "В" сохранить в памяти.

**Вариант 11.** Инициализируйте строку символов. Определить является ли данная строка правильным скобочным выражением (т.е. количество открывающих скобок равно количеству закрывающих)? Рассматривать только круглые скобки. Отсутствие скобок также трактуется как правильное скобочное выражение.

**Вариант 12.** Инициализируйте строку, содержащую часть символов в скобках. Удалить эту часть символьной строки вместе со скобками. Результирующую строку сохранить в памяти.

**Вариант 13.** Инициализируйте строку символов, среди которых есть одна открывающая и одна закрывающая скобка. Сохранить в памяти все символы, расположенные внутри этих скобок.

**Вариант 14.** Инициализируйте строку символов. Группы символов в ней, разделенные пробелом, считаются словами. Определить сколько слов начинаются на одну и ту же букву (букву задать). Количество слов сохранить в памяти.

**Вариант 15.** Инициализируйте строку символов, состоящую из отдельных слов, разделенных пробелами. В результирующей строке вставить вместо пробела запятую и пробел. Вывести на экран исходную и результирующую строки.

**Вариант 16.** В строке обменять местами самое длинное и самое короткое слово.

**В****ариа****нт 17.** Дан текст, состоящий из 6 слов по 8 символов, разделенных пробелом. Переставить слова в тексте так, чтобы каждое следующее слово начиналось с той буквы, на которую закончилось предыдущее. Первое слово оставить на месте.

**Ва****риант 18.** Дана последовательность латинских букв длиной 18 симво­лов. Расставить их по алфавиту.

**Вариант 19.** Дан текст 30 символов. Сжать текст, оставив между словами по одному пробелу.

**Вариант 20.** Дан текст 20 символов. Определить количество повторений сочетаний "ОР" в тексте и заменить его символом "!".

**Вариант 21.** Дан текст 35 символов. Определить количество слов, содержа­щих более 3-х символов. Слова разделяются одним пробелом.

**Вариант 22.** Дан текст 8 слов по 6 символов. В словах с четным номером изменить порядок букв на обратный.

**В****ариант 23.** Построить последовательность из n (n<=30) символов сле­дующего вида: АББВВВГГГГДДДДДЕЕЕЕЕЕ...

**Вариант 24.** Построить последовательность из 40 символов по заданной таблице, содержащей символ и количество его повторений. Например:

А - 3 раза

Б - 5 раз АААБББББСССССС

C - 6 раз

**В****ариа****нт 25.** Дан текст 32 символа, состоящий из слов, разделенных од­ним пробелом. Определить количество слов и количество букв в каждом слове.

**Ва****риант 26.** Дан текст 34 символа, состоящий из слов, разделенных од­ним пробелом. Определить количество слов, в которых буква Е встречается более чем 2 раза.

**Вариант 27.** Дана последовательность 37 символов. Определить частоту повторения каждого символа.

***ЗАДАНИЕ 2.***

1. Удалить из строки самое длинное слово. (Слова разделяются пробелами)

2. Удалить из строки первое слово (слова разделяются пробелами, запятыми, точкой)

3. Удалить из строки последнее слово (слова разделяются пробелами, запятой)

4. Удалить из строки слово с максимальным количеством букв «а»

5. Переставить во всех словах буквы в обратном порядке.

6. Удалить лишние пробелы из строки

7. Переставить в словах, длиной больше 3 символов, буквы в обратном порядке.

8. Удалить удвоенные буквы в словах.

9. Удалить слова с удвоенными буквами.

10. Переставить первое и последнее слово в строке.

11. Заменить в строке все буквы «а» на «b»

12. Удалить из строки буквы «b»

13. Заменить во всех слова, содержащих только цифры, цифры на пробел

14. Удалить все слова, содержащие только цифры.

15. удалить слова-палиндромы

16. Удалить слова, длиной больше 5 символов.

17. Удалить слова с повторяющимися буквами.

18. Удалить из строки слова, содержащие больше 2 букв «а»

19. Обменять самое длинное и самое короткое слова.

20. Обменять слова с максимальным и минимальным количеством букв «а»

21. Удалить слова, содержащие цифры.

22. Удалить слова, длиной меньше 4 символов

23. Удалить из строки слова, содержащие буквы «а».

24. Удалить слова, начинающиеся и заканчивающиеся на букву «а»

25. Заменить в каждом слове первую букву на заглавную

26. Вставить перед каждым числом (числа – слова, состоящие только из цифр) ноль.

27. Заменить в словах, длиной больше 4 символов, все буквы на одну букву «b»

**КОНТРОЛЬНЫЕ ВОПРОСЫ.**

1. Особенности выполнения изучаемых базовых команд процессора i8086/i8088.
2. Команды безусловного перехода.
3. Команды условного перехода. Организация разветвлений.
4. Разница в организации условных переходов для знаковых и беззнаковых данных.
5. Команды управления циклом.
6. Основные особенности работы с цепочечными командами: хранение строки-источника и строки-приемника, флаг направления, использование регистров SI и DI.
7. Целочисленный массив как строка в ассемблере.
8. Префикс повторения.
9. Понятие о байтах кода операции, способах адресации, регистрах и смещениях.
10. В следующем примере необходимо сравнить два массива целых чисел. В той точке, где массивы различаются необходимо поместить наименьший элемент массива в регистр AX. Проверьте и откорректируйте все логические ошибки:

***.data***

***list01 dw -1, 2, -4, 20***

***list02 dw -1, 2, 3, 20***

***.code***

***cld***

***lea si, list01***

***lea di, list02***

***mov cx,4***

***repe cmpsw***

***jl L1***

***dec si***

***mov ax, [si] ; переместить list01 в AX***

***jmp L2***

***L1: sub di,2***

***mov ax, [di] ; переместить list02 в AX***

***L2: ret***