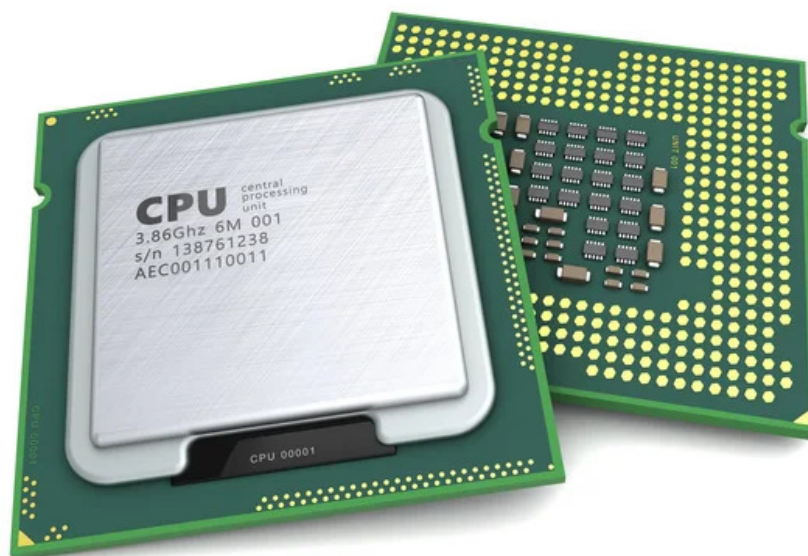


# **ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ASSEMBLER**

## **Цепочечные команды микропроцессора**

Практическое пособие

для студентов специальности  
1–40 01 01 «Программное обеспечение  
информационных технологий»



## Предисловие

Микропроцессор имеет достаточно мощные средства для реализации вычислительных операций. Для этого у него есть блок целочисленных операций и блок операций с плавающей точкой.

Система команд микропроцессора также имеет очень интересную группу команд, позволяющих производить действия над блоками элементов до 64 Кбайт или 4 Гбайт, в зависимости от установленной разрядности адреса `use16` или `use32`. Эти блоки логически могут представлять собой последовательности элементов с любыми значениями, хранящимися в памяти в виде двоичных кодов. Единственное ограничение состоит в том, что размеры элементов этих блоков памяти имеют фиксированный размер 8, 16 или 32 бита. Команды обработки строк предоставляют возможность выполнения семи операций-примитивов, обрабатывающих цепочки поэлементно.

## Цепочечные команды

*Цепочечные команды* также называют *командами обработки строк символов*. Под строкой символов понимается последовательность байт, а цепочка – это более общее название для случаев, когда элементы последовательности имеют размер слово или двойное слово. То есть цепочечные команды позволяют проводить действия над блоками памяти, представляющими собой последовательности элементов следующего размера: 8 бит (байт); 16 бит (слово); 32 бита (двойное слово).

Содержимое этих блоков для микропроцессора не имеет никакого значения. Это могут быть символы, числа и все что угодно. Главное, чтобы размерность элементов совпадала с одной из вышеперечисленных, и эти элементы находились в соседних ячейках памяти.

В системе команд микропроцессора имеются семь операций-примитивов обработки цепочек. Каждая из них реализуется в микропроцессоре тремя командами, в свою очередь, каждая из этих команд работает с соответствующим размером элемента – байтом, словом или двойным словом. Особенность всех цепочечных команд в том, что они, кроме обработки текущего элемента цепочки, осуществляют еще и автоматическое продвижение к следующему элементу данной цепочки.

Операции-примитивы и команды, с помощью которых они реализуются:

### 1. Пересылка цепочки:

`movs` `адрес_приемника`, `адрес_источника` (`MOVE String`) – переслать цепочку;

`movsb` (`MOVE String Byte`) – переслать цепочку байтов;

`movsw` (`MOVE String Word`) – переслать цепочку слов;

movsd (MOVE String Double word) – переслать цепочку двойных слов.

Команды производят копирование элементов из одной области памяти (цепочки) в другую.

## 2. Сравнение цепочек:

cmps адрес\_приемника, адрес\_источника (CoMPare String) – сравнить строки;

cmpsb (CoMPare String Byte) – сравнить строку байт;

cmpsw (CoMPare String Word) – сравнить строку слов;

cmpsd (CeMPare String Double word) – сравнить строку двойных слов.

Команды производят сравнение элементов цепочки-источника с элементами цепочки-приемника.

## 3. Сканирование цепочки:

scas адрес\_приемника (SCAning String) – сканировать цепочку;

scasb (SCAning String Byte) – сканировать цепочку байт;

scasw (SCAning String Word) – сканировать цепочку слов;

scasd (SCAning String Double Word) – сканировать цепочку двойных слов.

Команды производят поиск некоторого значения в области памяти.

## 4. Загрузка элемента из цепочки:

lods адрес\_источника (LOaD String) – загрузить элемент из цепочки в регистр-аккумулятор al/ax/eax;

lodsb (LOaD String Byte) – загрузить байт из цепочки в регистр al; lodsw (LOaD String Word) – загрузить слово из цепочки в ре-

гистр ax;

lodsd (LOaD String Double Word) – загрузить двойное слово из цепочки в регистр eax.

Эта операция позволяет извлечь элемент цепочки и поместить его в регистр-аккумулятор al, ax или eax.

## 5. Сохранение элемента в цепочке:

stos адрес\_приемника (STOre String) – сохранить элемент из регистра-аккумулятора al/ax/eax в цепочке;

stosb (STOre String Byte) – сохранить байт из регистра al в цепочке;

stosw (STOre String Word) – сохранить слово из регистра ax в цепочке;

stosd (STOre String Double Word) – сохранить двойное слово из регистра eax в цепочке.

Эта операция позволяет произвести действие, обратное команде lods, то есть сохранить значение из регистра-аккумулятора в элементе цепочки.

## 6. Получение элементов цепочки из порта ввода-вывода:

ins адрес\_приемника, номер\_порта (INput String) – ввести элементы из порта ввода-вывода в цепочку;

insb (INput String Byte) – ввести из порта цепочку байтов;

`insw` (INput String Word) – ввести из порта цепочку слов;  
`insd` (INput String Double Word) – ввести из порта цепочку двой-ных слов.

7. Вывод элементов цепочки в порт ввода-вывода:

`outs` номер\_порта, адрес\_источника (OUTput String) – вывести элементы из цепочки в порт ввода-вывода;

`outsb` (OUTput String Byte) – вывести цепочку байтов в порт вво-да-вывода;

`outsw` (OUTput String Word) – вывести цепочку слов в порт вво-да-вывода;

`outsd` (OUTput String Double Word) – вывести цепочку двойныхслов в порт ввода-вывода.

К цепочечным командам нужно отнести и префиксы повторения:

`rep`

`repe` или `repz`

`repne` или `repnz`

Префиксы повторения указываются перед нужной цепочечной командой в поле метки. Цепочечная команда без префикса выполняется один раз. Размещение префикса перед цепочечной командой заставляет ее выполняться в цикле. Отличия префиксов в том, на каком основании принимается решение о циклическом выполнении цепочечной команды: по состоянию регистра `ecx/cx` или по флагу нуля `zf`:

– префикс повторения `rep` (REPeat) используется с командами, реализующими операции-примитивы пересылки и сохранения элементов цепочек `movs` и `stos`. Префикс `rep` заставляет данные команды выполняться, пока содержимое в `ecx/cx` не станет равным 0. При этом цепочечная команда, перед которой стоит префикс, автоматически уменьшает содержимое `ecx/cx` на единицу. Та же команда, но без префикса, этого не делает;

– префиксы повторения `repe` или `repz` (REPeat while Equal or Zero) являются синонимами. Цепочечная команда выполняется до тех пор, пока содержимое `ecx/cx` не равно нулю или флаг `zf` равен 1. Как только одно из этих условий нарушается, управление передается следующей команде программы. Благодаря возможности анализа флага `zf` наиболее эффективно эти префиксы можно использовать с командами `cmps` и `scas` для поиска отличающихся элементов цепочек;

– префиксы повторения `repne` или `repnz` (REPeat while Not Equal or Zero) также являются синонимами. Префиксы `repne/repnz` заставляют цепочечную команду циклически выполняться до тех пор, пока содержимое `ecx/cx` не равно нулю или флаг `zf` равен нулю. При невыполнении одного из этих условий работа команды прекращается. Данные префиксы также можно использовать с командами `cmps` и `scas`, но для поиска совпадающих элементов цепочек.

*Особенность формирования физического адреса операндов ад-*

*рес\_источника* и *адрес\_приемника*. 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*. Допускается замена (с помощью префикса замены сегмента) только регистра *ds*, регистр *es* заменять нельзя. Вторые части адресов – смещения цепочек должны находиться:

- для цепочки-источника это регистр *esi/si* (Source Index register – индексный регистр источника);

- для цепочки-получателя это регистр *edi/di* (Destination Index register – индексный регистр приемника). Таким образом, полные физические адреса для операндов цепочечных команд следующие:

*адрес\_источника* – пара *ds:esi/si*;

*адрес\_приемника* – пара *es:edi/di*.

Команды *lds* и *les* позволяют получить полный указатель (сегмент:смещение) на ячейку памяти.

Семь групп команд, реализующих цепочечные операции-примитивы, имеют похожий по структуре набор команд. В каждом из этих наборов присутствуют одна команда с явным указанием операндов и три команды, не имеющие операндов. На самом деле набор команд микропроцессора имеет соответствующие машинные команды только для цепочечных команд ассемблера без операндов. Команды с операндами транслятор ассемблера использует только для определения типов операндов. После того как выяснен тип элементов цепочек по их описанию в памяти, генерируется одна из трех машинных команд для каждой из цепочечных операций. Поэтому все регистры, содержащие адреса цепочек, должны быть инициализированы заранее, в том числе и для команд, допускающих явное указание операндов. Так как цепочки адресуются однозначно, нет особого смысла применять команды с операндами. Главное – правильная загрузка регистров указателями.

Есть две возможности задания направления обработки цепочки: от начала цепочки к её концу, то есть в направлении возрастания адресов; от конца цепочки к началу, то есть в направлении убывания адресов.

Направление определяется значением флага направления *df* (Direction Flag) в регистре *eflags/flags*:

- если *df* = 0, то значения индексных регистров *esi/si* и *edi/di* будут автоматически увеличиваться (операция инкремента) цепочечными командами, то есть обработка будет осуществляться в направлении возрастания адресов;

- если *df* = 1, то значения индексных регистров *esi/si* и *edi/di* будут автоматически уменьшаться (операция декремента) цепочечными

ми командами, то есть обработка будет идти в направлении убывания адресов.

Состоянием флага `df` можно управлять с помощью двух команд, не имеющих операндов:

`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.

## Пересылка цепочек

Формат команды `movs`:

`movs` `адрес_приемника`, `адрес_источника`

Команда копирует байт, слово или двойное слово из цепочки, адресуемой операндом `адрес_источника`, в цепочку, адресуемую операндом `адрес_приемника`.

Команда `movs` пересылает только один элемент, исходя из его типа, и модифицирует значения регистров `esi/si` и `edi/di`. Если перед командой написать префикс `rep`, то одной командой можно переслать несколько элементов данных. Число пересылаемых элементов должно быть загружено в счетчик – регистры `cx` или `ecx`.

Порядок пересылки последовательности элементов из одной области памяти в другую с помощью команды `movs`. Этот набор действий можно рассматривать как типовой для выполнения любой цепочечной команды:

1. Установить значение флага `df` в зависимости от того, в каком направлении будут обрабатываться элементы цепочки – в направлении возрастания или убывания адресов.

2. Загрузить указатели на адреса цепочек в памяти в пары регистров `ds:(e)si` и `es:(e)di`.

3. Загрузить в регистр `ecx/cx` количество элементов, подлежащих обработке.

4. Записать команду `movs` с префиксом `rep`.

Пример 2.1. Пересылка символов из одной строки в другую. Строки находятся в одном сегменте памяти. Для пересылки используется команда-примитив `movs` с префиксом повторения `rep`.

```
masm
model small
.data
    source db 'Тестируемая строка$' ;строка-источник
    dest db 20 dup (' ') ;строка-приёмник
```

```

.stack
    db 256 dup (0)
.code
    assume ds:@data,es:@data
    main proc near
    mov ax,@data ;загрузка сегментных регистров
    mov ds,ax      ;настройка регистров ds и es
    mov es,ax      ;на адрес сегмента данных
    cld           ;сброс флага df – обработка строки от начала к концу
    lea si,source ;загрузка в si смещения строки-источника
    lea di,dest    ;загрузка в ds смещения строки-приёмника
    mov cx,20 ;для префикса rep счетчик повторений (длина строки)
rep movs dest,source ;пересылка строки
    lea dx,dest
    mov ah,09h      ;вывод на экран строки-приёмника
    int 21h
    mov ax,4c00h
    int 21h

    main endp
end main

```

Пересылка байт, слов и двойных слов производится командами `movsb`, `movsw` и `movsd`. Единственной отличительной особенностью этих команд от команды `movs` является то, что последняя может работать с элементами цепочек любого размера – 8, 16 или 32 бита. При трансляции команда `movs` преобразуется в одну из трех команд: `movsb`, `movsw` или `movsd`. В какую конкретно команду будет произведено преобразование, определяет транслятор исходя из размеров элементов цепочек, адреса которых указаны в качестве операндов команды `movs`. Адреса цепочек для любой из четырех команд должны формироваться заранее в регистрах `esi/si` и `edi/di`.

При использовании команды `movsb` изменится только строка с командой пересылки:

```

lea si,source ;загрузка в si смещения строки-источника
lea di,dest    ;загрузка в ds смещения строки-приёмника
mov cx,20 ;для префикса rep – счетчик повторений (длина строки)
rep movsb      ;пересылка строки

```

Отличие в том, что программа из примера 2.1 может работать с цепочками элементов любой из трех размерностей: 8, 16 или 32 бита, а последний фрагмент – только с цепочками байтов.

## Сравнение цепочек

Формат команды `cmps`:

`cmps` `адрес_приемника`, `адрес_источника`

`адрес_источника` определяет цепочку-источник в сегменте данных. Адрес цепочки должен быть заранее загружен в пару `ds:esi/si`; `адрес_приемника` определяет цепочку-приемник. Цепочка должна находиться в дополнительном сегменте, и ее адрес должен быть заранее загружен в пару `es:edi/di`.

Алгоритм работы команды `cmps` заключается в последовательном выполнении вычитания (элемент цепочки-источника – элемент цепочки-получателя) над очередными элементами обеих цепочек. Команда `cmps` производит вычитание элементов, не записывая при этом результат, и устанавливает флаги `zf`, `sf` и `of`. После выполнения вычитания очередных элементов цепочек командой `cmps` индексные регистры `esi/si` и `edi/di` автоматически изменяются в соответствии со значением флага `df` на значение, равное размеру элемента сравниваемых цепочек. Чтобы команда `cmps` выполнялась несколько раз, то есть производилось последовательное сравнение элементов цепочек, необходимо перед командой `cmps` определить префикс повторения.

С командой `cmps` можно использовать префиксы повторения `repe/repz` или `repne/repnz`:

- `repe` или `repz` – если необходимо организовать сравнение до тех пор, пока не будет выполнено одно из двух условий: достигнут конец цепочки (содержимое `ecx/cx` равно нулю); в цепочках встретились разные элементы (флаг `zf` стал равен нулю);

- `repne` или `repnz` – если нужно проводить сравнение до тех пор, пока не будет достигнут конец цепочки (содержимое `ecx/cx` равно нулю); в цепочках встретились одинаковые элементы (флаг `zf` стал равен единице).

Вместе с командой сравнения используется команда условного перехода `jcxz`. Ее работа заключается в анализе содержимого регистра `ecx/cx`, и если он равен нулю, то управление передается на метку, указанную в качестве операнда `jcxz`. Так как в регистре `ecx/cx` содержится счетчик повторений для цепочечной команды, имеющей любой из префиксов повторения, то, анализируя `ecx/cx`, можно определить причину выхода из заикливания цепочечной команды. Если значение в `ecx/cx` не равно нулю, то это означает, что выход произошел по причине совпадения либо несовпадения очередных элементов цепочек.

В таблице 2.1 представлены команды условной передачи управления, которые используются с командой сравнения `cmps` (для чисел



без знака).

Таблица 2.1 – Сочетание команд условной передачи управления с результатами команды `cmps` (для чисел без знака)

Причина прекращения операции сравнения	Команда условного перехода, реализующая переход
операнд_источник > операнд_приемника	<code>ja</code>
операнд_источник = операнду_приемнику	<code>jc</code>
операнд_источник <> операнду_приемнику	<code>jnc</code>
операнд_источник < операнда_приемника	<code>jb</code>
операнд_источник <= операнда_приемника	<code>jbe</code>
операнд_источник >= операнда_приемника	<code>jac</code>

Как определить местоположение очередных совпавших или несовпавших элементов в цепочках? После каждой итерации цепочечная команда автоматически осуществляет инкремент/декремент значения адреса в соответствующих индексных регистрах. Поэтому после выхода из цикла в этих регистрах будут находиться адреса элементов, находящихся в цепочке после (!) элементов, которые послужили причиной выхода из цикла. Для получения истинного адреса этих элементов

необходимо скорректировать содержимое индексных регистров, увеличив либо уменьшив значение в них на длину элемента цепочки.

Пример 2.2. Программа, которая сравнивает две строки, находящиеся в одном сегменте.

```
masm
model small
.data
    match db 0ah,0dh,'Строки совпадают$'
    failed db 0ah,0dh,'Строки не совпадают$'
    string1 db '0123456789',0ah,0dh,'$';исследуемые строки
    string2 db '0123406789$'
.stack
    db 256 dup (0)
.code
    assume ds:@data,es:@data ;привязка ds и es к сегменту данных
    main proc near
        mov ax,@data ;загрузка сегментных регистров
        mov ds,ax
        mov es,ax ;настройка es на ds
        ;вывод на экран исходных строк string1 и string2
```

```

    mov ah, 09h
    lea dx, string1
    int 21h
    lea dx, string2
    int 21h
;сброс флага df – сравнение в направлении возрастания адресов
    cld
    lea si, string1 ;загрузка в si смещения string1
    lea di, string2 ;загрузка в di смещения string2
    mov cx, 10 ;длина строки для префикса repe
;сравнение строк (пока сравниваемые элементы строк равны)
;выход при обнаружении не совпавшего элемента
    repe cmps string1, string2
    jcxz equal ;cx = 0, то есть строки совпадают
    jne not_match ;если не равны – переход на not_match
equal: ;иначе, если совпадают, то
    mov ah, 09h ;вывод сообщения
    lea dx, match
    int 21h
    jmp exit ;выход
not_match: ;не совпали
    mov ah, 09h
    lea dx, failed

    int 21h ;вывод сообщения
;теперь, чтобы обработать не совпавший элемент в строке,
;необходимо уменьшить значения регистров si и di
    dec si ;скорректировали адреса очередных элементов для получения
    dec di ;адресов несовпавших элементов
;сейчас в ds:si и es:di адреса несовпавших элементов
;здесь вставить код по обработке несовпавшего элемента
;после этого продолжить поиск в строке:
    inc si
;для просмотра оставшейся части строк необходимо установить
;указатели на следующие элементы строк за последними
;несовпавшими. После этого можно повторить весь процесс просмотра
;и обработки несовпавших элементов в оставшихся частях строк
    inc di
exit: ;выход
    mov ax, 4c00h
    int 21h
main endp

```

end main

Если сравниваются цепочки с элементами слов или двойных слов, то корректировать содержимое esi/si и edi/di нужно на 2 и 4 байта, соответственно.

## 2.3. Сканирование цепочек

Команды производят поиск некоторого значения в области памяти. Логически эта область памяти рассматривается как последовательность (цепочка) элементов фиксированной длины размером 8, 16 или 32 бита. Искомое значение предварительно должно быть помещено в регистр al/ax/eax. Выбор конкретного регистра из этих трех должен быть согласован с размером элементов цепочки, в которой осуществляется поиск.

Формат команды scas:

scas адрес\_приемника

Команда имеет один операнд, обозначающий местонахождение цепочки в дополнительном сегменте (адрес цепочки должен быть заранее сформирован в es:edi/di). Транслятор анализирует тип идентификатора адрес\_приемника, который обозначает цепочку в сегменте данных и формирует одну из трех машинных команд, scasb, scasw или scasd. Условие поиска для каждой из этих трех команд находится в строго определенном месте: если цепочка описана с помощью директивы db, то искомый элемент должен быть байтом и находиться в al, а сканирование цепочки осуществляется командой scasb; если

цепочка описана с помощью директивы dw, то это – слово в ax, и поиск ведется командой scasw; если цепочка описана с помощью директивы dd, то это двойное слово в eax, и поиск ведется командой scasd. Принцип поиска тот же, что и в команде сравнения cmps, то есть последовательное выполнение вычитания (содержимое\_регистра\_аккумулятора – содержимое\_очередного\_элемента\_цепочки). В зависимости от результатов вычитания производится установка флагов, при этом сами операнды не изменяются. С командой scas удобно использовать префиксы repe/repz или repne/repnz:

- repe или repz – если нужно организовать поиск до тех пор, пока не будет выполнено одно из двух условий: достигнут конец цепочки (содержимое ecx/cx равно 0); в цепочке встретился элемент, отличный от элемента в регистре al/ax/eax;

- repne или repnz – если нужно организовать поиск до тех пор, пока не будет выполнено одно из двух условий: достигнут конец цепочки (содержимое ecx/cx равно 0); в цепочке встретился элемент, совпадающий с элементом в регистре al/ax/eax.

Таким образом, команда `scas` с префиксом `repe/repz` позволяет найти элемент цепочки, отличающийся по значению от заданного в аккумуляторе. Команда `scas` с префиксом `repne/repnz` позволяет найти элемент цепочки, совпадающий по значению с элементом в аккумуляторе.

Пример 2.3. Поиск символа в строке.

```

masm
model small
.data
    find    db 0ah,0dh,'Символ найден! ','$'
    nofind  db 0ah,0dh,'Символ не найден.','$'
    mes     db 'String for search.',0ah,0dh,'$'
    nom     dw ? ;номер совпавшего символа должен быть 1310 = D16
.stack
    db 256 dup(0)
.code
    assume ds:@data,es:@data
    main proc near
        mov ax,@data
        mov ds,ax
        mov es,ax    ;настройка es на ds
        mov ah,09h
        lea dx,mes
        int 21h      ;вывод сообщения mes
        mov al,'a'   ;символ для поиска
        cld          ;сброс флага df

        lea di,mes ;загрузка в es:di смещения строки
        mov si,di   ;запоминаем адрес начала строки
        mov cx,18    ;для префикса repne длина строки
        ;поиск в строке (пока искомый символ и символ в строке не совпадут)
        ;выход при первом совпадении
        repne scas mes
        je found     ;если равны – переход на обработку,
        mov ah,09h    ;вывод сообщения о том, что символ не найден
        lea dx,nofind
        int 21h      ;вывод сообщения nofind
        jmp exit     ;переход на завершение программы
found:              ;совпали
        mov ah,09h
        lea dx,find
        int 21h      ;вывод сообщения find
    
```

```

;чтобы узнать место, где совпал элемент в строке,
;необходимо уменьшить значение в регистре di
    dec di
    sub di, si      ;находим номер совпавшего символа
    mov     nom, di
exit: mov ax, 4c00h
    int 21h
main endp
end main

```

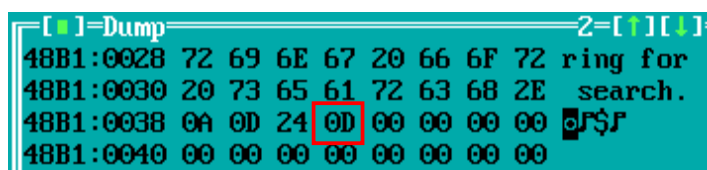


Рисунок 2.2 – Результат работы программы

## Загрузка элемента цепочки в аккумулятор

Эта операция-примитив позволяет извлечь элемент цепочки и поместить его в регистр-аккумулятор `al`, `ax` или `eax`. Эту операцию удобно использовать вместе с поиском (сканированием) с тем, чтобы, найдя нужный элемент, извлечь его (например, для изменения). Размер извлекаемого элемента определяется применяемой командой.

Формат команды `lods`:

`lods адрес_источника (LOaD String)` – загрузить элемент из цепочки в аккумулятор `al/ax/eax`.

Команда имеет один операнд, обозначающий строку в основном сегменте данных. Работа команды заключается в том, чтобы извлечь

элемент из цепочки по адресу, соответствующему содержимому пары регистров `ds:esi/si`, и поместить его в регистр `eax/ax/al`. При этом содержимое `esi/si` подвергается инкременту или декременту (в зависимости от состояния флага `df`) на значение, равное размеру элемента. Эту команду удобно использовать после команды `scas`, локализующей местоположение искомого элемента в цепочке.

**Пример 2.4.** Программа сравнивает командой `cmps` две цепочки байт в памяти `string1` и `string2` и помещает первый несовпавший байт из `string2` в регистр `al`, номер несовпавшего символа – в переменную `nom`. Для загрузки этого байта в регистр-аккумулятор `al` используется команда `lods`. Префикса повторения в команде `lods` нет, так как он просто не нужен.

masm

```

model small
.data
    string1 db 'String one.',0ah,0dh,'$'
    string2 db 'String two.',0ah,0dh,'$'
    mes_eq   db 'The strings are equal',0ah,0dh,'$'
    find     db 'Not equal in the al register',0ah,0dh,'$'
    nom      dw ? ;номер несовпавшего элемента
.stack
    db 256 dup (0)
.code
    assume ds:@data,es:@data
    main proc near
        mov ax,@data;загрузка сегментных регистров
        mov ds,ax
        mov es,ax    ;настройка es на ds
        mov ah,09h
        lea dx,string1
        int 21h      ;вывод string1
        lea dx,string2
        int 21h      ;вывод string2
        cld          ;сброс флага df
        lea di,string1 ;загрузка в es:di смещения строки string1
        lea si,string2 ;загрузка в ds:si смещения строки string2
        mov nom,si
        mov cx,11    ;для префикса repe – длина строки
        ;поиск в строке (пока нужный символ и символ в строке не равны)
        ;выход – при первом несовпадении
        repe cmps string1,string2
        jcxz eql      ;если равны – переход на eql
        jmp no_eq     ;если не равны – переход на no_eq
    eql: mov ah,09h   ;выводим сообщение о совпадении строк
        lea dx,mes_eq
        int 21h      ;вывод сообщения mes_eq
        jmp exit     ;переход на конец
    no_eq: mov ah,09h ;обработка несовпадения элементов
        lea dx,find
        int 21h      ;вывод сообщения find
        ;чтобы извлечь несовпавший элемент из строки
        ;в регистр-аккумулятор, уменьшаем значение регистра si и тем самым
        ;перемещаемся к действительной позиции элемента в строке
        dec si        ;команда lods использует ds:si-адресацию
                     ;ds:si указывает на позицию в string2
        lods string2 ;загрузим элемент из строки в al, это символ 't'

```

```

sub si,nom
dec si
mov nom,si ;номер несовпавшего символа = 7
exit: mov ax,4c00h
int 21h
main endp
end main

```

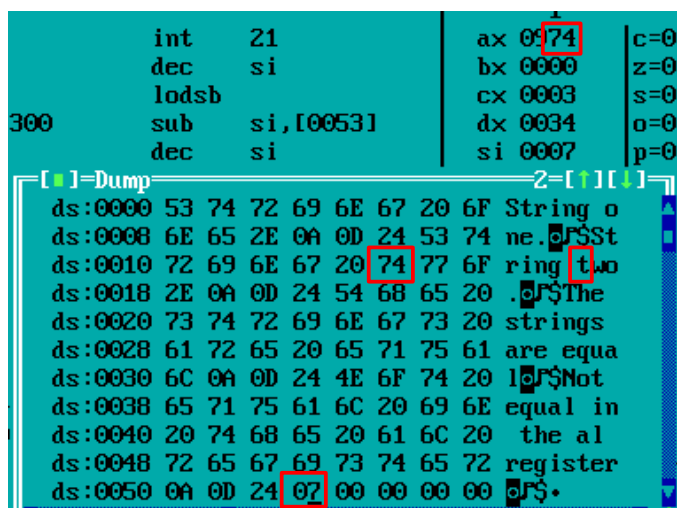


Рисунок 2.3 – Результат выполнения программы

## Перенос элемента из аккумулятора в цепочку

Эта операция-примитив позволяет сохранить значение из регистра-аккумулятора в элементе цепочки. Операцию удобно использовать вместе с операциями поиска (сканирования) `scans` и загрузки `lods` с тем, чтобы, найдя нужный элемент, извлечь его в регистр и записать на его место новое значение.

Формат команды `stos`:

`stos адрес_приемника (STOrage String)` – сохранить элемент из регистра-аккумулятора `al/ax/eax` в цепочке.

Команда имеет один операнд `адрес_приемника`, адресуящий цепочку в дополнительном сегменте данных. Команда пересылает элемент из аккумулятора (регистра `eax/ax/al`) в элемент цепочки по адресу, соответствующему содержимому пары регистров `es:edi/di`. При этом содержимое `edi/di` подвергается инкременту или декременту (в зависимости от состояния флага `df`) на значение, равное размеру элемента цепочки.

Пример 2.5. Программа производит замену в строке всех символов 'а' на другой символ, вводимый с клавиатуры.

masm

```

model small
.data
    find db 'Character is found',0ah,0dh,'$'
    nochar db 'Character not found',0ah,0dh,'$'
    mes1 db 'Source string:',0ah,0dh,'$'
    string db 'Search character in this string',
            0ah,0dh,'$' ;строка для поиска
    mes2 db 'Enter character-->$'
    mes3 db 0ah,0dh,'New string: ',0ah,0dh,'$'
.stack
    db 256 dup(0)
.code
    assume ds:@data,es:@data    ;привязка ds и es
                                ;к сегменту данных

    main proc near
    mov ax,@data                ;загрузка сегментных регистров
    mov ds,ax
    mov es,ax                    ;настройка es на ds
                                mov ax,0002h    ;очистка экрана
                                int 10h
    mov ah,09h
    lea dx,mes1
    int 21h ;вывод сообщения mes1
    lea dx,string
    int 21h ;вывод string
    mov al,'a' ;символ для поиска
    cld ;сброс флага df
    lea di,string ;загрузка в di смещения string
    mov cx,31 ;для префикса repne - длина строки
;поиск в строке string до тех пор, пока символ в al и очередной
;символ в строке не равны: выход - при первом совпадении
    repne scas string
    je found ;если элемент найден, то переход на found
                ;иначе, если не найден, то вывод сообщения nochar
    mov ah,09h
    lea dx,nochar
    int 21h
    jmp exit ;переход на выход
found: mov ah,09h
    lea dx,find
    int 21h ;вывод сообщения об обнаружении символа
    mov ah,09h ;ввод символа с клавиатуры

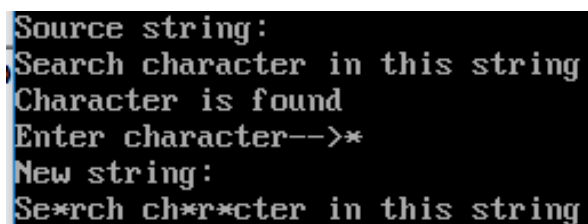
```



```

    lea dx,mes2
    int 21h      ;вывод сообщения mes2
    mov ah,01h
    int 21h      ;в al – введенный символ
    mov bl,al    ;сохраняем введенный символ
;устанавливаем начальные данные для замены
    cld          ;сброс флага df
    lea di,string ;загрузка в di смещения string
    mov cx,31    ;для префикса repne – длина строки
;поиск в строке string до тех пор, пока символ в al и очередной
;символ в строке не равны: выход – при первом совпадении
cycl: mov al,'a' ;символ для поиска
    repne scas string
    jne exit    ;если элемент не найден, переход на конец программы
;корректируем di для получения значения действительной позиции
;совпавшего элемента в строке и регистре al
    dec di
new_char:      ;блок замены символа
    mov al,bl
    stos string ;сохраним введенный символ (из al) в строке
                ;string в позиции старого символа
;переход на поиск следующего символа 'a' в строке
    inc di      ;указатель в строке string на следующий,
                ;после совпавшего, символ
    jmp cycl    ;на продолжение просмотра string
exit: mov ah,09h
    lea dx,mes3
    int 21h      ;вывод сообщения mes3
    lea dx,string
    int 21h      ;вывод сообщения string
    mov ax,4c00h
    int 21h
main endp
end main

```



```

Source string:
Search character in this string
Character is found
Enter character-->*
New string:
Se*rch ch*r*cter in this string

```

Рисунок 2.4 – Результат работы программы

## **Ввод элемента цепочки из порта ввода-вывода**

Данная операция вводит цепочки элементов из порта ввода-вывода и реализуется командой `ins`, имеющей следующий формат:

`ins` адрес\_приемника, номер\_порта (Input String) – ввести элементы из порта ввода-вывода в цепочку.

Команда вводит элемент из порта, номер которого находится в регистре `dx`, в элемент цепочки, адрес которого определяется операндом `адрес_приемника`. Адрес цепочки должен быть явно сформирован в паре регистров `es:edi/di`. Размер элементов цепочки должен быть согласован с размерностью порта; он определяется директивой резервирования памяти, с помощью которой выделяется память для размещения элементов цепочки. После пересылки команда `ins` производит коррекцию содержимого `edi/di` на величину, равную размеру элемента, участвовавшего в операции пересылки, при этом учитывается состояние флага `df`.

## **Вывод элемента цепочки в порт ввода-вывода**

Данная операция выводит элементов цепочки в порт ввода-вывода. Она реализуется командой `outs`, имеющей следующий формат:

`outs` номер\_порта, адрес\_источника (Output String) – вывести элементы из цепочки в порт ввода-вывода.

Эта команда выводит элемент цепочки в порт, номер которого находится в регистре `dx`. Адрес элемента цепочки определяется операндом `адрес_источника`. Адрес цепочки должен быть явно сформирован в паре регистров `ds:esi/si`. Размер структурных элементов цепочки должен быть согласован с размерностью порта. Он определяется директивой резервирования памяти, с помощью которой выделяется память для размещения элементов цепочки. После пересылки команда `outs` производит коррекцию содержимого `esi/si` на величину, равную размеру элемента цепочки, участвовавшего в операции пересылки, при этом учитывается состояние флага `df`.

## Литература

1. Гук, М. Процессоры Pentium III, Athlon и другие / М. Гук, В. Юров. – СПб. : Питер, 2000. – 379 с.
2. Зубков, С. В. Ассемблер для DOS, Windows и UNIX / С. В. Зубков. – М. : ДМК Пресс, 2000. – 534 с.
3. Программирование на языке ассемблера для персональных ЭВМ : учебное пособие / А. Ф. Каморников [и др.]. – Гомель : ГГУ, 1995. – 95 с.
4. Пустоваров, В. И. Ассемблер : программирование и анализ корректности машинных программ / В. И. Пустоваров. – К. : Издательская группа BHV, 2000. – 480 с.
5. Сван, Т. Освоение Turbo Assembler / Т. Сван. – Киев : Диалектика, 1996. – 540 с.
6. Юров, В. Assembler / В. Юров. – СПб. : Питер, 2001. – 624 с.
7. Юров, В. Assembler : практикум / В. Юров. – СПб. : Питер, 2002. – 400 с.
8. Юров, В. Assembler : специальный справочник / В. Юров. – СПб.: Питер, 2000. – 496 с.

