Практика 6: Операции со строками.

Строки и операции со строками уже рассматривались ранее. Точнее, ранее рассмотрены были способы задания строк и были приведены примеры кода для сравнения строк и для поиска и замены символов в строке. Заметим что для работы со строками были использованы стандартные команды работы с памятью, однако в языке ассемблера есть и специальные команды для работы со строками:

- **MOVS**
- **CMPS**
- **SCAS**
- **LODS**
- **STOS**

Далее рассмотрим эти команды подробнее, однако сначала рассмотрим еще три команды, непосредственно связанные с обработкой строк.

Префиксы повторения.

Для организации цикла некоторые команды могут использовать специальные инструкции :

- **REP**
- **REPE/REPZ**
- **REPNE/REPNZ**

Команду — а правильнее сказать префикс повторения - REP проще всего сравнить в командой LOOP — команда действует как цикл, беря количество повторов из регистра CX или ECX. Перед использованием префикса необходимо поместить в CX/ECX число повторений команды. Префикс повторения автоматически уменьшает регистр CX/ECX и повторяет выполняемую команду до тех пор, пока CX/ECX не будет равен нулю. Отличие от команды LOOP в том, что префикс действует только на одну команду, причем далеко не каждую команду можно использовать с префиксом.

Префикс	Команда
REP	**MOVS**
	LODS
	STOS
REPE/REPZ **REPNE/REPNZ**	**CMPS**
	SCAS

В отличии от REP, префиксы REPE/REPZ и REPNE/REPNZ прекращают повторение команды не только при достижении нулевого значения регистра CX или ECX, но и в зависимости от состояния флага ZF:

- REPE/REPZ может повторяться лишь до тех пор, пока ZF=1
- REPNE/REPNZ — пока ZF=0.

Операции над строками.

Вернемся к командам работы со строками.

MOVS используется для копирования одной строки в другую. На адрес источника указывает пара регистров DS:ESI, а на приемник пара ES:EDI, но в большинстве случаев можно считать, что на источник указывает ESI, а на приёмник регистр EDI.
Напомним, что ранее строки рассматривались в виде непрерывной последовательности байт, интерпретируемых как символы таблицы ASCII. Это верно,правда требует уточнения — строки могут рассматриваться и как последовательности двух или четырех байтовых значений.
Фактически, команды MOVS нет в процессоре. При появлении этой инструкции в программе, компилятор определяет размерность операнда и на основании этого вызывает соответствующую ей инструкцию :

- **MOVSB** – источник и приёмник это строка из символов в 1 байт.
- **MOVBW** – строки из символов по 2 байта
- **MOVSD** – строки из символов по 4 байта

В программе можно как явно вызывать одну из трёх инструкций, так и использовать «обезличенную» инструкцию MOVS.
После копирования символа происходит декремент (при DF = 1) или инкремент (при DF = 0) содержимого регистров ESI и EDI. Если команда оперирует байтами, индексный регистр изменяется на 1, в других случаях на 2 или 4. Операция

повторяется количество раз, заданное в CX/ECX и останавливается при достижении 0.
Для установки флага DF можно воспользоваться командами

- **CLD** — DF=0, то есть обработку строки следует начинать с первого символа
- **STD** — DF=1, то есть обработку строки следует начинать с последнего символа

Пример:

копирование из строки в строку

Решение:

```
.data
text1str DB `My STRING 1`,0
text2str DB `              `,0
.code
...
cld
mov esi,text1str
mov edi,text2str
mov ecx,12
rep movsb
push STD_OUTPUT_HANDLE
call [GetStdHandle]
push 0
push inputnum
push 12
push text2str
push eax
call [WriteConsole]
```

...

Задание:

заменить в предыдущем примере команду MOVSB на MOVSW. Чему в этом случае должно быть равно ECX и почему.

Решение:

ответ 6, причину предлагается объяснить самостоятельно.

LODS используется для чтения символа из строки по адресу ESI. Символ помещается в AL/AX/EAX в зависимости от размера операнда и – соответственно – выбора команды

- **LODSB**
- **LODSW**
- **LODSD**

STOS используется для заполнения строки выбранным символом. Символ хранится в AL/AX/EAX, а на строку указывает регистр EDI. Аналогично предыдущей команде в зависимости от размера элемента строки вызывается команда **STOSB** , **STOSW** или **STOSD** .

CMPS используется для сравнения строк. Первый операнд берется из регистра ESI а второй из EDI. Фактически, над символами строк выполняется операция CMP, которая выставляет значение флага ZF. Префиксы REPE/REPZ используются для поиска отличающихся элементов строк, а REPNE/REPNZ для поиска совпадающих элементов. Команда имеет варианты

- **CMPSB**
- **CMPSW**

- **CMPSD**

SCAS для сравнения элемента строки с заданным символом. Символ берется из регистра AL/AX/EAX в зависимости от типа, объявленного при создании строки. На текущий символ строки указывает пара регистров EDI. Фактически к символу и элементу строки применяется команда CMP и далее формируется значение флага ZF. Префиксы REPE/REPZ используются для поиска элемента строки отличающегося от заданного, а REPNE/REPNZ для поиска совпадающих элементов.

Далее рассмотрим некоторые способы работы со строками.

Сравнение строк.

Для сравнения двух строк используем **CMPS** по следующему алгоритму :

1. Выбираем способ сравнения – с первого байта (CLD) или с последнего (STD)
2. Записываем операнды в регистры
3. Задаем в ECX количество сравниваемых элементов
4. Вызываем CMPS в цикле
5. Оцениваем результат

Вариант сравнения выглядит так:

```
.data
    text1str DB `My STRING 1`,0
    text2str DB `My STRING 2`,0
    msg1str db 'Strings are equal.',10,13,0
```

```
    msg2str db 'Strings are not equal.',10,13,0
    inputnum DD 0
    stdoutputhandle dd 0
.code
  start:
<..>
        push STD_OUTPUT_HANDLE
        call [GetStdHandle]
        mov [stdoutputhandle],eax
       cld
       mov esi,text1str
       mov edi,test2str
       mov ecx,12
       repe cmpsb
       je strequal
        push 0
        push inputnum
        push 24
        push msg2str
        push [stdoutputhandle]
        call [WriteConsole]
        jmp strnotequal
       strequal:
        push 0
        push inputnum
        push 20
        push msg1str
        push [stdoutputhandle]
        call [WriteConsole]
       strnotequal:
       ...
```

Осталось обсудить, как узнать длину строки. Здесь всё достаточно просто – надо посчитать количество байт до нулевого завершающего символа.

Задание :

Написать код подсчёта длины строки

Решение (один из очевидных, но не самых быстрых вариантов):

```
    mov    ecx,text1str
    xor    eax, eax
    dec    eax
notzero:
    inc    eax
    cmp    byte[ecx+eax], 0
    jne    notzero
```

Результат окажется в EAX.

Таким образом, цикл сравнения строк чуть увеличится:

1. Вычисляем длины строк и сравниваем их. Если не равны, значит строки различны, иначе длину одной из них сохраняем в стеке
2. Выбираем способ сравнения
3. Записываем операнды в регистры
4. Задаем в ECX количество сравниваемых элементов, восстановив длину из стека.
5. Вызываем CMPSB в цикле
6. Оцениваем результат

Подсчет количества слов в строке.

Фактически это задача поиска символа разделителя в строке. Символом разделителем обычно является символ пробела. Вариант решения задачи может быть следующий

```
    cld
    mov edi,text1str
    mov al,' '
    mov ecx,12
    xor edx,edx
loopcalc:
    repne scasb
    inc edx
    test ecx,ecx
    jne loopcalc
```

Помещаем в AL символ пробела; в EDI помещаем адрес переменной со строкой. В ECX добавим длину строки и очистим EDX. Теперь ставим метку и запускаем scanb с префиксом. Эта команда остановится при первом найденом пробеле. Увеличим EDX, проверим не кончилась ли строка и - если нет – перейдем к метке, тем самым продолжив поиск.

Задание:

Переписать код подсчета длины строки с использованием scasb.

Решение:

```
    cld
    mov edi,text1str
    mov al,0
    xor ecx,ecx
```

```
dec ecx
repne scasb
mov edx,ecx
xor ecx,ecx
dec ecx
sub ecx,edx
```

Приведённый выше код подсчёта слов в строке не учитывает многие особенности и может выдавать неверный результат. Например

- несколько пробелов подряд в данном коде будет выдавать не разделение двух слов, а «засчитывать» каждый пробел за 1 слово
- начальные и конечные пробелы так же будут увеличивать счётчик слов

В качестве домашнего задания предлагается улучшить приведенный код и избавить его от указанных недостатков.