Весна 2021

Системное программное обеспечение

Онлайн-лекции

Лекция №5: **Работа с массивами. Строковые команды**

Доцент, к.т.н. ГОЛЬЦОВ Александр Геннадьевич

Команда LEA

- "Load effective address"
- Второй аргумент - ячейка памяти, адрес которой задан в любом формате, в том числе косвенном
- Загружает в первый аргумент адрес (смещение) ячейки памяти
- lea ax, s ➔ есть такая команда, но ее код 4-байтовый и она эквивалентна трехбайтовой mov ax, offset x
- lea ax, [bx+si+75] ➔ сложение сразу трех чисел!

Команда XLAT

- Выборка байта из массива по индексу ("преобразование по таблице")
- Без аргументов (жестко использует AL и BX)

mem(ds:[BX+AL]) ➔ AL

- Можно подменить сегмент по умолчанию: xlat es:

Формирование шестнадцатеричной записи числа


```
.286
.data
HEX     db '01234567890ABCDEF'
OutStr  db 4 dup(?),'$'
N       dw 1234h
.code
.....................
        mov     bx, offset HEX

        mov     al,byte ptr N+1
        shr     al,4
        xlat
        mov     OutStr[0],al

        mov     al,byte ptr N+1
        and     al,0Fh
        xlat
        mov     OutStr[1],al

        mov     al,byte ptr N
        shr     al,4
        xlat
        mov     OutStr[2],al

        mov     al,byte ptr N
        and     al,0Fh
        xlat
        mov     OutStr[3],al

        mov     ah,9
        mov     dx,offset OutStr
        int 21h
        .....................
```

Если не написать .286, то

shr al,4 ➔

```
shr al,1
shr al,1
shr al,1
shr al,1
```

Сложение многобайтных чисел

```
A       dd 1
B       dd 2
SUM     dd ?
.........................
mov     ax, word ptr A
add     ax, word ptr B
mov     word ptr SUM, ax

mov     ax, word ptr A+2
adc     ax, word ptr B+2
mov     word ptr SUM+2, ax
```

```
Mas     dw 1,2,3,4,5
SUM     dd ?
.........................
        xor     bx,bx
        mov     word ptr Sum,bx
        mov     word ptr Sum+2, bx

        mov     cx,5

m1:     mov     ax,mas[bx]
        add     bx,2
        add     word ptr SUM,ax
        adc     word ptr SUM+2, 0

        loop m1
```

Проверка 4-байтового числа на равенство нулю


```
.data
p   dd 1,2,0,3,0,5
.....................
.code
.....................
    mov bx, offset p

    mov ax, word ptr [bx]
    or    ax,word ptr [bx+2]
    jz    ZeroElement
```

Условные переходы

- JCXZ - переход, если CX=0
- После **cmp <операнд1>, <операнд2>**:
  - JG - числа со знаком, переход, если первый операнд больше второго (анализирует знак и переполнение: SF=OF, ZF=0)
  - см. также JL, JGE, JLE
  - JA - беззнаковые числа, переход, если первый операнд больше второго (анализирует перенос: CF=0, ZF=0)
  - см. также JB, JAE, JBE
- JZ=JE, JNZ=JNE - переход, если ноль (если сравниваемые числа равны, zero)
- JC, JNC - переход, если сформирован перенос (carry)
- JS, JNS - переход, если старший бит результата 1 (знак отрицательный, sign)
- JNP=JPO, JP=JPE - четность битов младшего байта результата (parity even/odd)

$$OF = CF \text{ xor } C_{6\text{-}7}$$

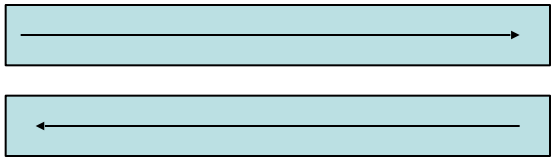
$$OF = CF \text{ xor } C_{14\text{-}15}$$

Строковые команды

- Адрес элемента обрабатываемого массива задан парой регистров:
  DS:SI – если массив – источник данных
  ES:DI – если массив – приемник данных
- Всегда держите в голове эти регистры именно парами, не разбивая их! Никогда не говорите, что адрес источника в SI, а приемника в DI, забыв упомянуть сегментные регистры!
- SI = source index, DI = destination index
- Элементы массива - байты или слова (в 386+ – также двойные слова)
- Строковая команда что-то делает с элементом массива и изменяет индекс для перехода к следующему элементу
- Важно значение флага DF
- Строковые команды не имеют параметров
- Могут повторяться CX раз или исполняться одиночно

Флаг DF

- Задает направление перебора элементов массива строковыми командами
- DF=0: перебор в сторону увеличения адресов
- DF=1: перебор в сторону уменьшения адресов
- SI и/или DI увеличиваются или уменьшаются на размер элемента массива (1 или 2)
- Нет значения этого флага по умолчанию! При входе в программу его следует считать не определенным!
- Сбросить: CLD
- Установить: STD



Команда LODS (LODSB, LODSW)

- LOaD String element – для последовательной загрузки элементов массива в аккумулятор
- DS:SI содержит адрес элемента массива
- Нет смысла использовать кратно (CX раз подряд)

- LODSB - работает с байтами, LODSW - со словами:

AL ← byte ptr [DS:SI] или
if not DF
    SI ← SI+1
else
    SI ← SI-1

AX ← word ptr [DS:SI]
if not DF
    SI ← SI+2
else
    SI ← SI-2

Команда STOS (STOSB, STOSW)

- STOre String element – для сохранения аккумулятора в качестве очередного элемента массива
- ES:DI содержит адрес элемента массива
- При кратном использовании позволяет заполнить массив начальным значением (обычно – нулями)

- STOSB - работает с байтами, STOSW - со словами:

AL ➔ byte ptr [ES:DI] или

if not DF

DI ← DI+1

else

DI ← DI-1

AX ➔ word ptr [ES:DI]

if not DF

DI ← DI+2

else

DI ← DI-2

Совместное использование LODS и STOS

Часто удобен такой алгоритм перебора массива:

загрузить в аккумулятор (LODS)
преобразовать или проверить условие
сохранить (или не сохранять) (STOS)

Префиксы повторения

- Перед строковой командой в машинном коде и в мнемонике может стоять префикс повторения:
  - REP - повторять CX раз
  - REPE/REPZ - повторять CX раз пока "равно" (флаг нуля взведен, если сбросится - повторение прекратится)
  - REPNE/REPNZ - повторять CX раз пока "не равно" (флаг нуля сброшен, если взведется - повторение прекратится)
- CX уменьшается, затем проверяется на достижение 0; начальное CX=0 означает 65536 повторений
- Условные префиксы обычно применяются с командами, которые сравнивают элементы массива с чем-то, это сравнение отражается в изменении флагов, в т.ч. ZF
- Уменьшение CX в ходе повторения не влияет на флаги

Заполнение массива

```
.data
mas       dw 10 dup(?)
.....................................
.code
          mov ax,@data
          mov ds,ax
.....................................
          push ds
          pop es
          mov di, offset mas ; ES:DI -> mas
          mov ax, 1024 ; значение для заполнения
          mov cx, 10

          cld         ; флаг направления сбросить явно!
rep       stosw
```

Копирование ненулевых слов

```
.data
len         equ 10
wa          dw 1,0,2,0,3,0,4,0,5,6
wb          dw len dup(-1) ; принимает данные
.....................
.code
.....................
            push ds ; DS стандартным образом был привязан к @data
            pop es

            mov si, offset wa
            mov di, offset wb
            cld         ; сбросить флаг направления явно!

            mov cx,len

m1:         lodsw       ; прочесть байт в AX, SI++
            test        ax,ax       ; сформировать флаг 0, не разрушая (AND)
            jz          skip
            stosw       ; записать слово из AX, DI=DI+2
skip:       loop m1
```

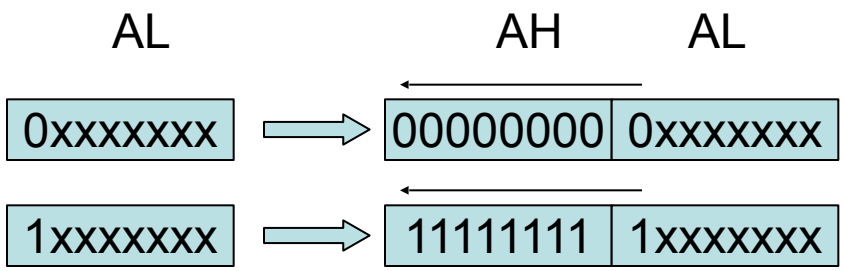
Конвертация массива байтов в массив слов

```
.data
len         equ 10
ba          db 1,2,3,4,5,-1,-2,-3,-4,-5 ; источник
wa          dw len dup(?) ; принимает данные
.....................
.code
.....................
            push ds ; DS стандартным образом был привязан к @data
            pop es

            mov si, offset ba
            mov di, offset wa
            cld         ; сбросить флаг направления явно!

            mov cx,len

m1:         lodsb       ; прочесть байт в AL, SI++
            cbw         ; размножить знаковый бит AL в AH
            stosw       ; записать слово из AX, DI=DI+2
            loop m1
```

Команды CBW и CWD

- Нужны для преобразования знакового значения меньшей разрядности в значение большей разрядности
- CBW:

  AL ➔ AX, знаковый бит AL заполняет AH



- CWD:

  AX ➔ DX:AX, знаковый бит AX заполняет DX

Команда MOVS (MOVSB, MOVSW)

- MOVe String element – для пересылки элемента одного массива в элемент другого массива
- DS:SI - адрес элемента массива-источника
  ES:DI - адрес элемента массива-приемника
- При кратном использовании позволяет скопировать сразу область памяти размером до 64к
  (movsw эффективнее)

- MOVSB - работает с байтами, MOVSW - со словами:

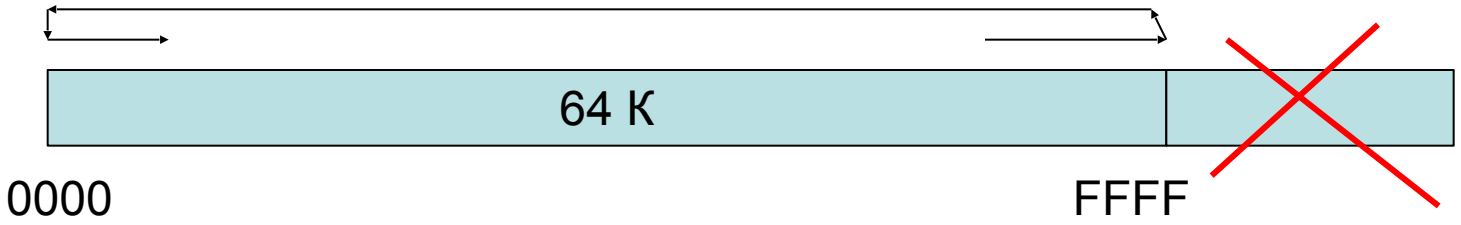
byte ptr [DS:SI] ➔ byte ptr [ES:DI] или word ptr [DS:SI] ➔ word ptr [ES:DI]

if not DF
    SI ← SI+1, DI ← DI+1
    DI+2
else
    SI ← SI-1, DI ← DI-1

if not DF
    SI ← SI+2, DI ←
else
    SI ← SI-2, DI ← DI-2

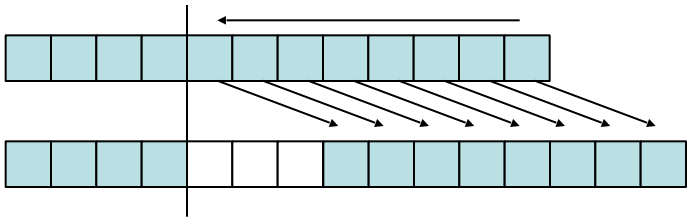
Достижение границы сегмента

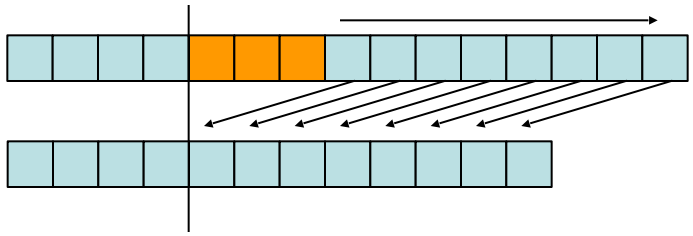
- В ходе выполнения строковой команды с префиксом повторения значение SI или DI может увеличиться после FFFF или уменьшиться после 0000
- При этом сегмент не изменится, и обработка данных продолжится в районе "противоположной границы" того же сегмента.



Особенности пересылки областей памяти

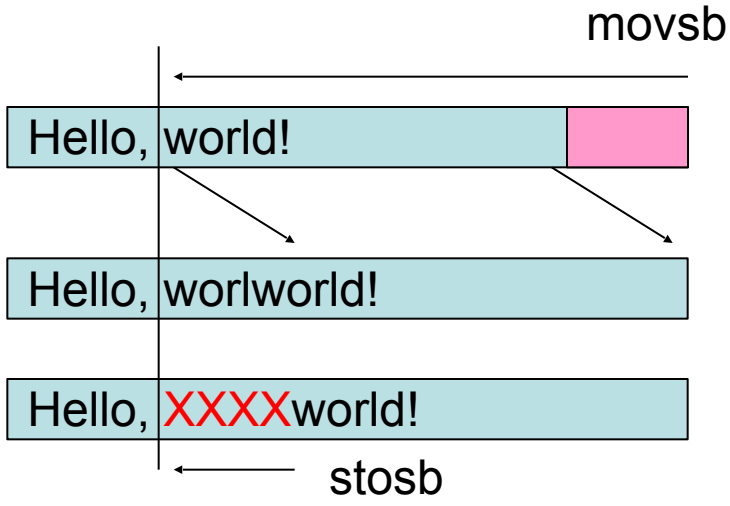
- Если области памяти источника и приемника не пересекаются - вопросов не возникает
- Часто области источника и приемника пересекаются. Например, нужно раздвинуть массив, чтобы вставить элементы внутрь, или удалить элементы из середины
- Если раздвигаем элементы - нужно перебирать элементы от конца области к началу,
  если удаляем кусок массива - от начала к концу.





Раздвинуть строку

```
.data
s           db 'Hello, world!', 25 dup(' ')
len         equ $-s
.code
            mov ax, @data
            mov ds,ax
            mov es, ax
.....................................
            std          ; будем начинать с конца строки
            mov di,offset s + len-1
            mov si, offset s + len-1-4
            mov cx, len-7-4  ; вставим 4 символа после 'Hello, '
rep         movsb

            mov cx,4
            mov al, 'X'
rep         stosb
```



Команда SCAS (SCASB, SCASW)

- SCAn String element – для поиска значения аккумулятора в массиве
- Выполняет неразрушающее вычитание элемента массива из аккумулятора (аналог команды CMP), формирует флаги
- ES:DI - адрес элемента массива (неочевидно!)
- При кратном использовании с условным префиксом позволяет проводить поиск значения в массиве

- SCASB - работает с байтами, SCASW - со словами:

```
cmp AL,byte ptr [ES:DI]      или      cmp AX, word ptr [ES:DI]
if not DF                             if not DF
    DI ← DI+1                             DI ← DI+2
else                                  else
    DI ← DI-1                             DI ← DI–2
```

Вычисление длины строки

```
.data
s       db 'Hello, world',0,25 dup (?)  ; 0 - признак конца
slen    db ?

.code
        mov ax, @data
        mov ds, ax
        mov es, ax
        mov di,offset s
        cld
        mov al,0            ; считаем, что 0 в конце строки есть обязательно!
        mov cx, 1000        ; заведомо большое значение
repne   scasb

        sub cx,1000-1       ; мы нашли 0 в позиции 1000-CX
        neg cx
        mov slen,cl
```

А если в выделенной строке укажем не 0, а пробел - найдем позицию пробела. Но важно знать длину строки - пробела ведь может не быть.

Поиск пробела

```
.data
s           db 12,'Hello, world' ; первый байт равен длине строки
SpacePos db 0     ; позиция пробела, считая с 1, 0=не найдено

.code
            mov ax, @data
            mov ds,ax
            mov es, ax
            mov di,offset s+1
            cld
            xor cx,cx
            mov cl,s              ; CX содержит длину, не забыть иниц. DS!
            jcxz        goout
            mov al,' '            ; ищем пробел
repne       scasb
            jne         goout     ; если не нашли пробел

            sub cl,s              ; мы нашли 0 в позиции len-CX
            neg cl
            mov SpacePos,cl
goout:      ........................
```

Команда CMPS (CMPSB, CMPSW)

- CoMPare String element – для сравнения элементов двух массивов
- Выполняет неразрушающее вычитание элемента массива-приемника из элемента массива-источника (аналог команды CMP), формирует флаги
- DS:SI - адрес элемента массива-источника
  ES:DI - адрес элемента массива-приемника
- Один из способов применения - сравнение строк (в том числе на больше/меньше!) и поиск подстроки в строке

- CMPSB - работает с байтами, CMPSW - со словами:

CMP byte ptr [DS:SI], byte ptr [ES:DI]
if not DF
    SI ← SI+1, DI ← DI+1
    DI+2
else
    SI ← SI-1, DI ← DI-1
    DI-2

CMP word ptr [DS:SI],word ptr [ES:DI]
if not DF
    SI ← SI+2, DI ←
else
    SI ← SI-2, DI ←

Поиск подстроки в строке

- S - строка, LS - длина строки,
  SS - искомая подстрока, LSS - длина подстроки
  пусть нумерация с 1
  результат: 0 (не найдено) или позиция подстроки
- i = 1 (начальная позиция в S),
  результат = 0
- Повторять LS – LSS + 1 раз:
  - Сравнить массив, начиная с S_i, с массивом SS на длину LSS
  - Если сравнение прошло, то i является искомой позицией, иначе
  - Увеличить i и продолжить цикл

Эффективность строковых команд

- Использование строковых команд вместо эквивалентных им обычных, как правило, делает код короче.
- В ряде случаев необходимые подготовительные действия (настройка нужных регистров) могут свести на нет это преимущество.
- Часто без применения строковых команд можно закодировать алгоритм проще и быстрее.
- По скорости работы строковые команды могут различаться на разных моделях процессоров, нужно смотреть справочник по машинным циклам в составе тех или иных команд.

Указатели. Команды LDS и LES

- Дальние указатели в i8086 хранятся в памяти в формате: сначала смещение, потом сегментная часть:

  2345:0012h ➔ 12 00 45 23

- Хранимый в памяти дальний указатель можно загрузить в сегментный (DS или ES) + произвольный регистр одной командой:

  LDS/LES<регистр>, [<адрес>]

- Например:

```
a db 12
pt dd a          ; указатель на a
.....................
lds si, pt       ; ds:si указывает на a
```

Умножение

- Команды MUL и IMUL
- MUL <8-битный операнд>
  AL * <операнд> ➔ AX
- MUL <16-битный операнд>
  AX * <операнд> ➔ DX:AX
- MUL - трактует оба операнда как числа без знака
  IMUL - как числа со знаком в доп. коде
- Операнд может быть регистром или ссылкой на ячейку памяти.

Умножение на степень двойки

- Используется сдвиг влево:

$$X * 2^n = X \text{ shl } n$$

- Правило действует в том числе для отрицательных чисел – просто сдвигаем дополнительный код
- Просто умножить на два - сложить число с самим собой

Деление

- Команды DIV и IDIV
- DIV <8-битный операнд>
  AX / <операнд> ➔ частное в AL, остаток в AH
- DIV <16-битный операнд>
  DX:AX / <операнд> ➔ частное в AX, остаток в DX
- DIV - трактует оба операнда как числа без знака
  IDIV - как числа со знаком в доп. коде
- Операнд может быть регистром или ссылкой на ячейку памяти.
- При делении на 0 или при получении частного, не умещающегося в половинную разрядность, будет ошибка (аварийное завершение программы)
- Делитель (байт или слово) должен быть по модулю больше, чем старшая половина делимого (слова или двойного слова)

Деление на степень двойки

- Для положительных чисел используется сдвиг вправо:

$$X / 2^n = X \text{ shr } n$$

- Для отрицательных чисел допустимо применять арифметический сдвиг (для положительных sar = shr):

$$X / 2^n = X \text{ sar } n$$

но в этом случае округление пойдет не к 0, а в меньшую сторону, что верно с математической т.з., но часто не годится в алгоритме

-1 sar 1 = -1 -2 sar 1 = -1
-3 sar 1 = -2 -4 sar 1 = -2

Смысл замены умножения и деления сдвигами

- 8086:
  деление - до 180 тактов
  умножение - до 150 тактов
  сдвиг регистра - 2 такта
  сдвиг ячейки памяти - 16+ тактов
- Современные процессоры персональных компьютеров способны выдавать результаты умножения, деления и сдвигов в примерно одинаковом темпе - каждый такт.
- В микроконтроллерах может вообще не быть операций умножения и деления.

Лаб 4

- Ничего не вводим с клавиатуры! Описываем данные средствами языка ассемблера.
- Результатом может быть преобразованная исходная строка или другая строка, получаемая в ходе работы с исходной.
- Вывести исходную и результирующую строки на экран (возможно - посимвольно).
- Отделять обработку от вывода: сначала сформировать полностью строку-результат, а потом - вывести ее на экран целиком

Спасибо за внимание.