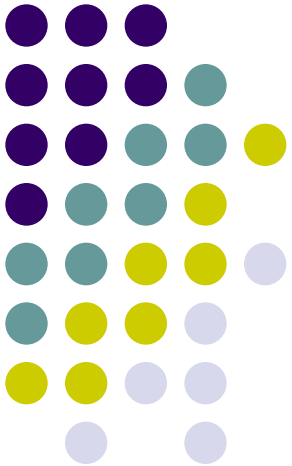
# Тема 7

# Работа с массивами



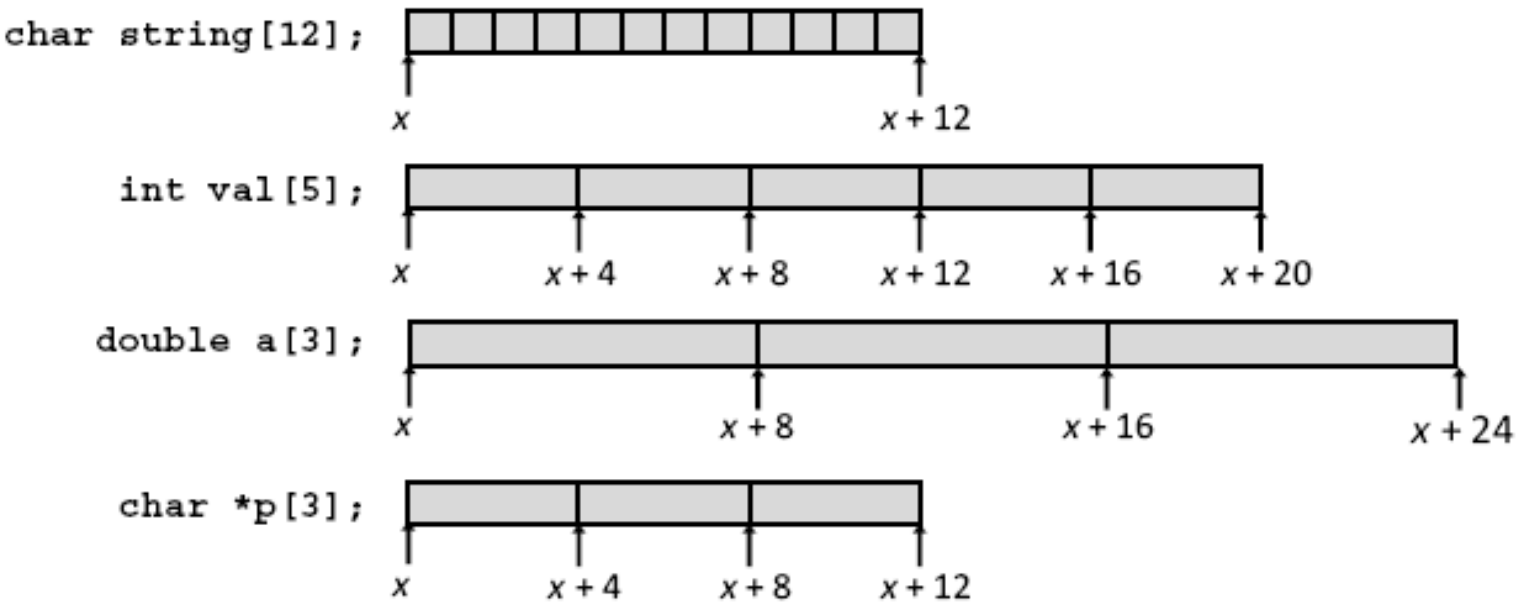
# Работа с одномерными массивами

- Массивы – размещение в памяти

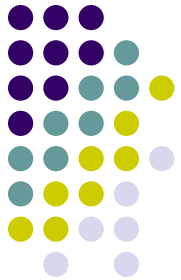
  `T A[L];`

  – Массив элементов типа T, размер массива – L

  – Массив располагается в непрерывном блоке памяти размером *L* * `sizeof` (*T*) байт



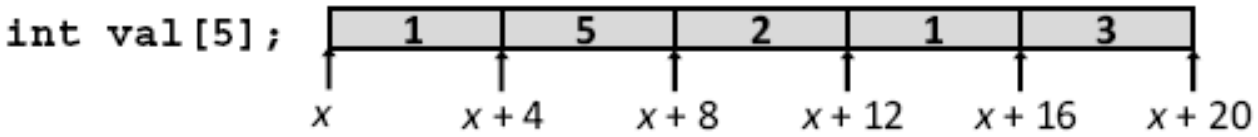
# Работа с одномерными массивами



- Доступ к элементам массива

```
T   A[L];
```

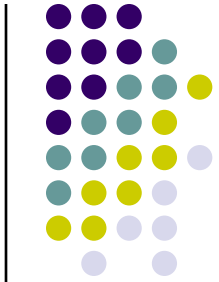
  - Массив элементов типа T, размер массива – L
  - Идентификатор A может использоваться как указатель на элемент массива с индексом 0. Тип указателя – T*




- Ссылка — Тип — Значение

| | | |
|---|---|---|
| `val[4]` | `int` | 3 |
| `val` | `int *` | *x* |
| `val+1` | `int *` | *x* + 4 |
| `&val[2]` | `int *` | *x* + 8 |
| `val[5]` | `int` | ?? |
| `*(val+1)` | `int` | 5 |
| `val + i` | `int *` | *x* + 4 *i* |

# Работа с одномерными массивами

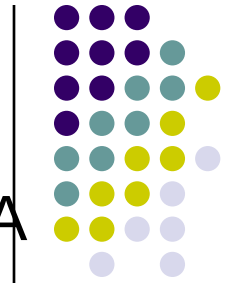


Для доступа к элементам массива с использованием механизма индексации необходимо хранить индекс в регистре общего назначения. При этом возможны варианты:

- в регистре хранится смещение в байтах относительно начала массива;
- в регистре хранится смещение в байтах относительно начала сегмента;
- в регистре хранится индекс, а смещение рассчитывается при обращении к массиву.

Для доступа к элементу массива допускается использовать два регистра.

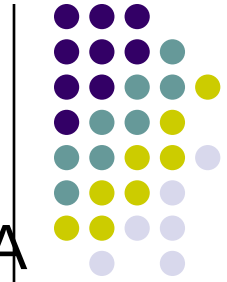
# Работа со статическими массивами (пример 1)



Задача: найти сумму первых N элементов массива A

```
void main() {
    int A[] = {21, 4, 32, 45, 67, -21, 34, 56, 21, 3};
    int N = 10, rezt;
   _asm {
        mov    ecx, N
        xor    ebx, ebx     //смещение относительно
                            //начала массива
        xor    eax, eax
cycle:
        add    eax, A[ebx] //базовая адресация
        add    ebx, 4
        dec    ecx
        cmp    ecx, 0
        jne    cycle
        mov    rezt, eax
   }
   cout <<  rezt << "\n"; }
```

# Работа со статическими массивами (пример 2)



Задача: найти сумму первых N элементов массива A

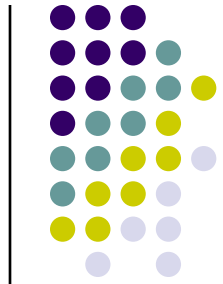
```
void main() {
    int A[] = {21, 4, 32, 45, 67, -21, 34, 56, 21, 3};
   int N = 10, rezt;
   _asm {
       mov   ecx, N
       xor   ebx, ebx //индекс элемента массива
       xor   eax, eax
cycle:
       add   eax, A[4*ebx]
       inc   ebx
       dec   ecx
       cmp   ecx, 0
       jne   cycle
       mov   rezt, eax
   }
   cout <<  rezt << "\n";
}
```

# Работа со статическими массивами (пример 3)

Задача: найти сумму первых N элементов массива A

```
void main() {
  int A[] = {21, 4, 32, 45, 67, -21, 34, 56, 21, 3};
   int N = 10, rezt;
  _asm {
      mov   ecx, N
      lea   ebx, A       //смещение относительно
                         // начала сегмента
      xor   eax, eax
cycle:
      add   eax, [ebx]
      add   ebx, 4
      dec   ecx
      cmp   ecx, 0
      jne   cycle
      mov   rezt, eax
  }
  cout <<  rezt << "\n"; }
```

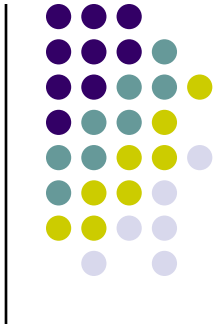
# Работа со статическими массивами(пример 4)



Задача: найти сумму первых N элементов массива A

```
void main() {
    int A[] = {21, 4, 32, 45, 67, -21, 34, 56, 21, 3};
    int N = 10, rezt;
    _asm {
      mov   ecx, N
      lea   ebx, A //смещение относительно начала сегмента
      xor   eax, eax
      xor   edi, edi  //смещение относительно
                      //начала массива
cycle:
      add   eax, [ebx][edi] //базово-индексная адресация
      add   edi, 4
      dec   ecx
      cmp   ecx, 0
      jne   cycle
      mov   rezt, eax
      }
      cout <<  rezt << "\n"; }
```

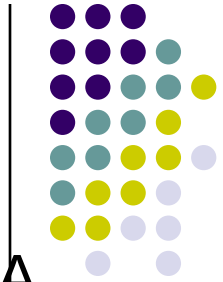
# Работа с динамическими массивами (пример 5)



Задача: найти сумму первых N элементов массива A

```
void main() {
    int *A = new int[10]; //заполнение массива A
    int N = 10, rezt;
    _asm {
       mov   ecx, N
       mov   ebx, A //смещение относительно начала сегмента
       xor   eax, eax
       xor   edi, edi    //смещение относительно начала
                         //массива
cycle:
       add   eax, [ebx][edi]
       add   edi, 4
       dec   ecx
       cmp   ecx, 0
       jne   cycle
       mov   rezt, eax
       }
     cout <<  rezt << "\n";}
```

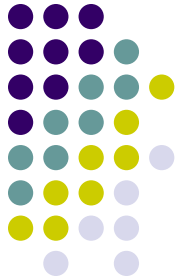
# Работа с динамическими массивами (пример 6)



Задача: найти сумму первых N элементов массива A

```
void main() {
   int *A = new int[10]; //заполнение массива A
   int N = 10, rezt;
   _asm {
        mov   ecx, N
        mov   ebx, A       //смещение относительно начала
                                  //сегмента
        xor   eax, eax
cycle:
        add   eax, [ebx][4*ecx-4]
        loop  cycle
end_cycle:
        mov   rezt, eax
   }
   cout <<  rezt << "\n";
}
```

# Работа с массивами (пример 7)



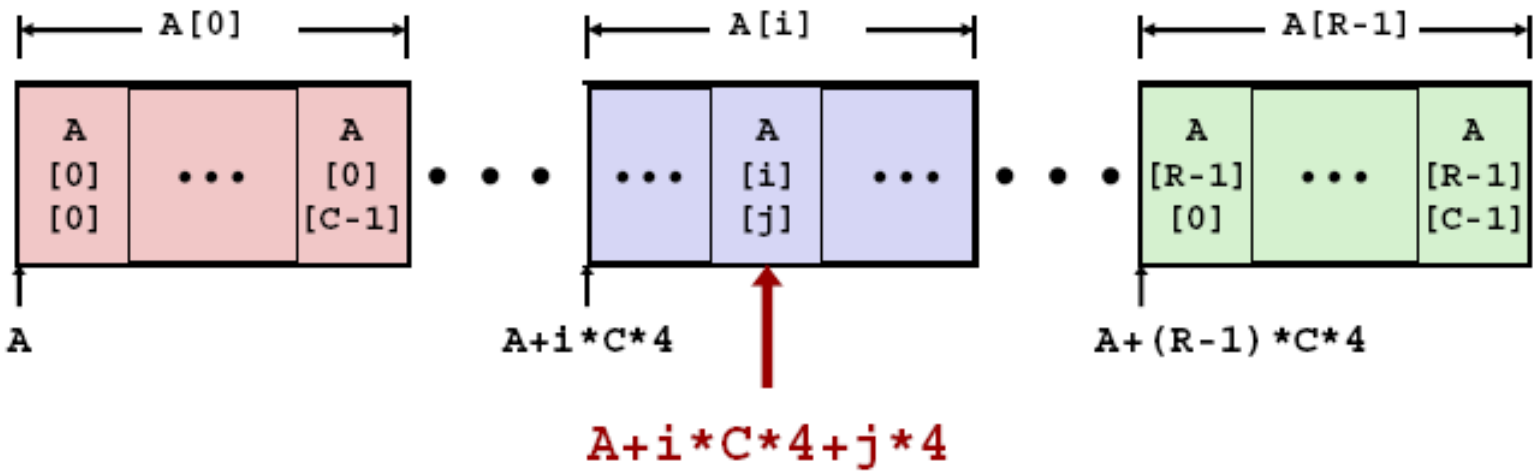
Задача: найти количество элементов беззнакового массива A из N чисел, стоящих перед первым нулем

```
void main() {
   unsigned A[10]; //заполнение массива A
    int N = 10, rezt;
   _asm {
       mov   ecx, N
       xor   ebx, ebx
       xor   eax, eax
cycle:cmp    A[ebx], 0
       je    m1
       inc   eax
       add   ebx, 4
m1:    loopne       cycle
       mov   rezt, eax
   }
   cout <<  rezt << "\n";}
```

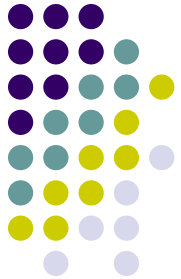
# Работа с двумерными массивами

- Элементы массива
  - **A[i][j]** элемент типа *T*, который требует *K* байт
  - Адрес элемента **A** + *i* * (*C* * *K*) + *j* * *K* = *A* + (*i* * *C* + *j*)* *K*

```
int A[R][C];
```



# Работа с двумерными массивами (пример)



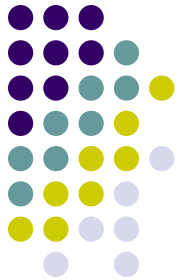
Задача: сколько раз число встречается в матрице (базово-индексная адресация с маштабированием)

```
void main() {
     short aray[5][2]= {{1,2},{3,4},{5,6},{7,3},{9,0}};
     short elem=3,          //элемент для поиска
           oundtime=0; //количество
   __asm
   {
       push esi
       xor eax,eax
       lea   ebx, aray    // bx = строки в матрице
       mov   ecx,5 // число для внешнего цикла (по строкам)
external:          //внешний цикл по строкам
```

# Работа с двумерными массивами (пример)

```
    //сохранение в стеке счётчика внешнего цикла
        push ecx
   //число для внутреннего цикла (по столбцам)
        mov  ecx,2
        mov   esi,0 // esi = столбцы в матрице
iternal:            //внутренний цикл по столбцам в строке
   // в ax 1-й элемент матрицы
        mov   ax,word ptr [ebx][esi*2]
// сравниваем содержимое текущего элемента в ax
// с искомым элементом:
        cmp ax, elem
// если текущий не совпал с искомым,
// то переход на next для продолжения поиска
        jne   next
```

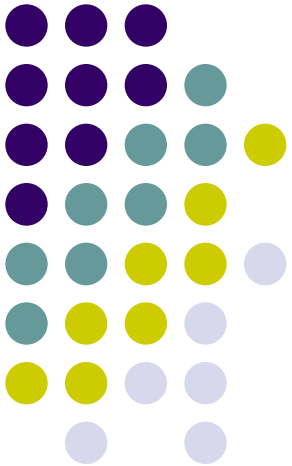
# Работа с двумерными массивами (пример)



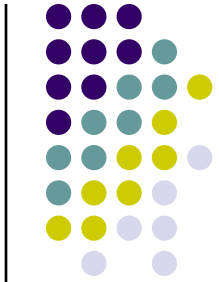
```
    // иначе увеличиваем счётчик совпавших
        inc   foundtime
next: // передвижение на следующий элемент в строке
        inc   esi
        loop  iternal // цикл по строке cx=2 раз
//продвижение в матрице
        pop   ecx     //восстанавливаем CX из стека (5)
        add   ebx, 4  //передвигаемся на следующую строку
        loop  external // цикл (внешний)
        pop esi
    }
    if (foundtime)
        cout << "Такой элемент в массиве присутствует " <<
    foundtime << " раз " << endl;
    else cout << "Нет такого элемента в массиве!" << endl;
}
```

# Тема 8

# Строковые команды



# Команды для работы со строками

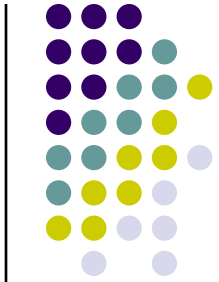


Команды для работы со строками (строковые команды) не обладают, по сравнению с ранее рассмотренными командами, особыми функциональными возможностями.

Преимущества использования таких команд:

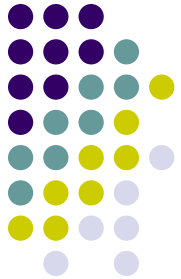
- более короткая и понятная запись исходного кода;
- более быстрая работа за счет того, что не теряется время на междукомандные переходы;
- возможность работать в режиме «память – память».

# Термины, используемые в строковых командах



- *Строка*, или *цепочка* – последовательность байтов, слов или двойных слов (элементов строки), расположенная в смежных участках памяти.
- *Источник* и *приемник* – две строки, с которыми работают строковые команды. Источник не изменяется, а приемник может меняться. Источник находится в сегменте, связанном с регистром DS, а приемник – с регистром ES.
- *Текущий элемент* строки – элемент, адрес которого находится в регистре SI/ESI (для источника) или в регистре DI/EDI (для приемника).

# Термины, используемые в строковых командах (продолжение)



- *Аккумулятор* – регистр AL, AX или EAX (в зависимости от длины элемента  строки).
- *Направление* просмотра строки определяется флагом DF (если он установлен, строка просматривается от больших адресов к меньшим).

# Фазы выполнения строковых команд

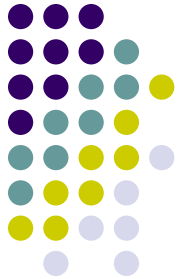
- обработка текущего элемента строки в соответствии с командой;
- переход к следующему элементу строки.

# Мнемоника строковых команд

| Код команды | Действие, выполняемое командой |
|---|---|
| **MOVS [x]** | пересылает текущий элемент строки-источника в строку-приемник |
| **CMPS [x]** | сравнивает текущие элементы строки-приемника и строки-источника, устанавливая соответствующие флаги |
| **LODS [x]** | пересылает текущий элемент строки-источника в регистр-аккумулятор |
| **STOS [x]** | пересылает содержимое регистра-аккумулятора в текущий элемент строки-приемника |
| **SCAS [x]** | сравнивает содержимое регистра-аккумулятора с текущим элементом строки-приемника |
| **CLD** | сбрасывает флаг DF |
| **STD** | устанавливает флаг DF |

x – B, W или D (в зависимости от длины элемента строки)

# Две формы записи строковых команд (на примере MOVS)



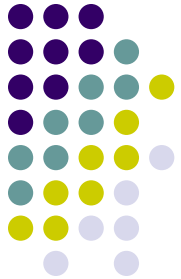
- Первая форма

```
MOVS приемник, источник
```

- Вторая форма (без операндов)

```
MOVSB       ; цепочка байтов
MOVSW       ; цепочка слов
MOVSD       ; цепочка двойных слов
```

# Префиксы повторения в строковых командах



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

Префиксы повторения записываются непосредственно перед командой, в той же строке:

```
REPE CMPB
```

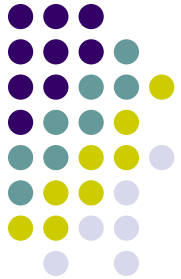
Префиксы работают с регистром CX/ECX, используя его как счетчик количества повторений

# Шаги выполнения строковых команд с префиксом повторения

На каждом этапе цикла выполняются следующие действия:

- проверка ECX. Если он равен 0 – выход из цикла и переход к следующей команде;
- обработка текущего элемента строки в соответствии с командой;
- уменьшение ECX на единицу без изменения значения флага (декремент);
- проверка флага ZF, если префиксом является REPE или REPNE: выход из цикла, если префиксом является REPE и ZF=0 (последнее сравнение не совпало) или используется префикс REPNE и ZF=1 (последнее сравнение совпало);
- изменение значения индексных регистров в соответствии со значением флага направления и переход на начало цикла.

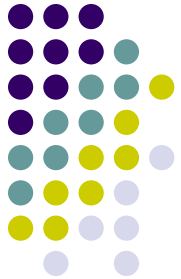
# Пример 1 использования строковых команд



**Задача:** переслать содержимое одной строки в другую

```
int main() {
  char s1[20], s2[20];
  strcpy(s1, "Hello, world!");
   _asm {
       mov   ecx, 20
       lea   edi, s2
       lea   esi, s1
       rep   movsb
   }
   cout << s1 <<"\n";
   cout << s2 <<"\n";
   return 0;
}
```

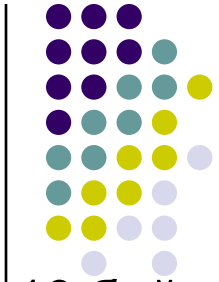
# Пример 2 использования строковых команд



**Задача:** Заполнить массив из 256 байтов последовательно символами с кодами от 255 до 0.

```
int main() {
  char s1[256];
  _asm {
      cld
      lea   edi, s1
      mov   ecx, 256
      mov   al, 0ffh
m1:
      stosb
      dec   al
      loop  m1
  }
  cout << s1 <<"\n";
  return 0;
}
```

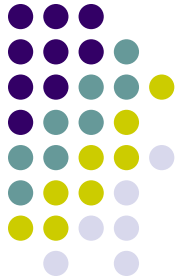
# Пример 3 использования строковых команд



**Задача:** Выполнить циклический сдвиг элементов массива из 10 байт, т.е. переслать первый элемент на место второго, второй – на место третьего, …, последний – на место первого.

```
void main() {
  char s1[] = "0123456789";
  _asm {
      std    ; пересылку выполняем с конца массива
      lea    edi, s1
      add    edi, 9 ; edi указывает на последний элемент
      mov    esi, edi
      dec    esi        ;  а esi – на предпоследний
      mov    ah, es:[edi] ; последний элемент сначала
                          ; сохраняем
      mov    ecx, 9
  rep movsb               ; пересылка
      mov    es:[edi], ah ; на первое место записываем
                          ; сохраненное
  }}
```

# Пример 4 использования строковых команд



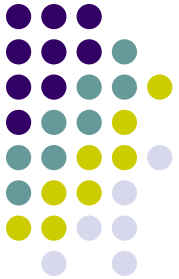
**Задача:** Подсчитать количество пробелов в строке

```
int main() {
  char s1[21];
  char rezt;
  strcpy(s1, "    0123b b vbbv  b ");
  _asm {
      cld
      xor   ah, ah ; здесь будет накапливаться результат
      mov   al, ' '
      lea   edi, s1
      mov   ecx, 20
```

# Пример 4 использования строковых команд (продолжение)

```
m1:
   repne      scasb
        jne    m2
        inc    ah
        jmp    m1
m2:
        mov    rezt, ah
   }
   cout << s1 <<"\n";
   cout << (int)rezt <<"\n";
   return 0;

}
```



# Спасибо за внимание!