© 2016 МГУ/ВМК/СП

Лекция 0xA

12 марта

© 2016 МГУ/ВМК/СП

Две обратные задачи

```
f:
   ...
   mov   edx, dword [ebp + 8]
   movsx ax, byte [a + edx]
   mov   word [b + edx + edx], ax
   ...
```

```
____________ a[N];
____________ b[N];

void f(________ i) {
   _______________;
}
```

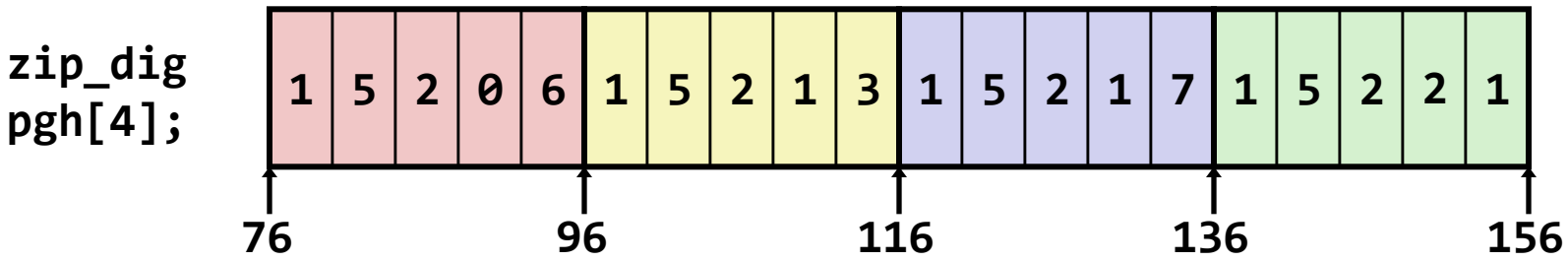
```
g:
   ...
   mov   edx, dword [ebp + 8]
   movzx eax, word [c + edx + edx]
   mov   byte [d + edx], al
   ...
```

```
____________ c[N];
____________ d[N];

void g(________ i) {
   _______________;
}
```

© 2016 МГУ/ВМК/СП

```
#define PCOUNT 4
zip_dig pgh[PCOUNT] =
  {{1, 5, 2, 0, 6},
   {1, 5, 2, 1, 3 },
   {1, 5, 2, 1, 7 },
   {1, 5, 2, 2, 1 }};
```



- “`zip_dig pgh[4]`” эквивалентно “`int pgh[4][5]`”
  - Переменная **pgh**: массив из 4 элементов, расположенных непрерывно в памяти
  - Каждый элемент – массив из 5 **`int`**’ов, расположенных непрерывно в памяти
- Всегда развертывание по строкам (Row-Major)

© 2016 МГУ/ВМК/СП

- Объявление

  T **A**[*R*][*C*];

  – 2D массив элементов типа *T*

  – *R* строк, *C* столбцов

  – Размер типа *T* – *K* байт

- Размер массива

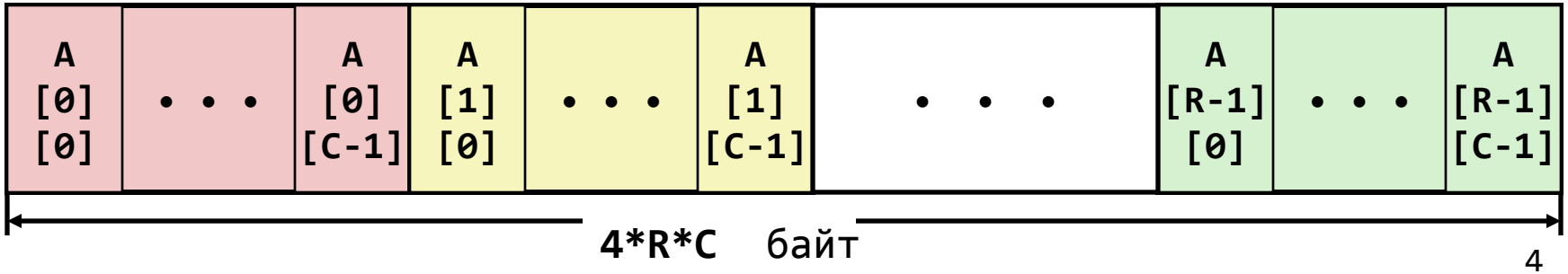
  – R * C * K байт

- Размещение в памяти

  – Развертывание по строкам

$$\begin{bmatrix} \texttt{A[0][0]} & \cdots & \texttt{A[0][C-1]} \\ \vdots & & \vdots \\ \texttt{A[R-1][0]} & \cdots & \texttt{A[R-1][C-1]} \end{bmatrix}$$

int A[R][C];

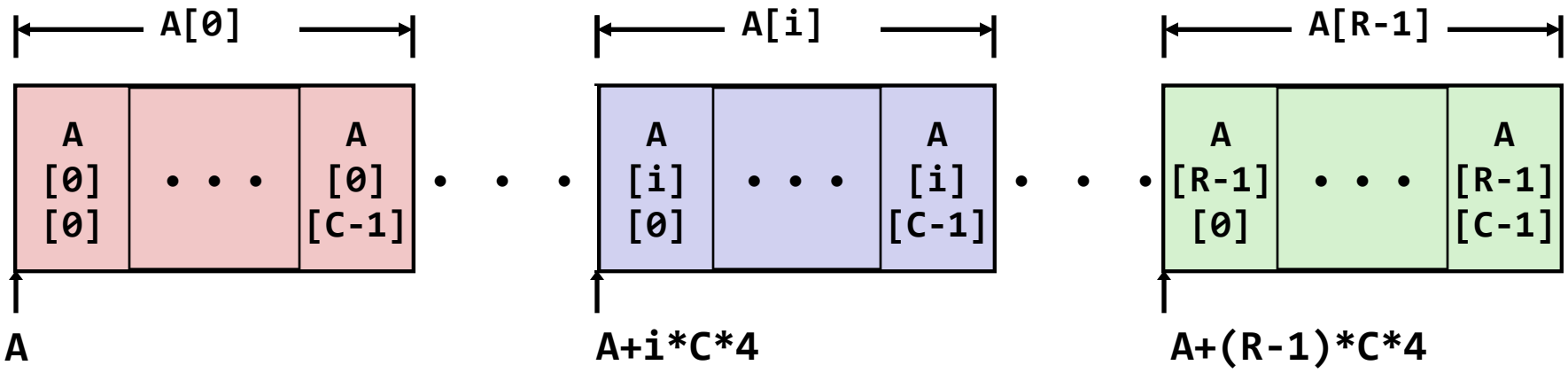


© 2016 МГУ/ВМК/СП

- Доступ к строкам
  - **A[i]** массив из *C* элементов
  - Каждый элемент типа *T* требует *K* байт
  - Начальный адрес строки с индексом i

A + $i * (C * K)$

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



© 2016 МГУ/ВМК/СП

```
int *get_pgh_zip(int index){
  return pgh[index];
}
```

```
#define PCOUNT 4
zip_dig pgh[PCOUNT] =
  {{1, 5, 2, 0, 6},
   {1, 5, 2, 1, 3 },
   {1, 5, 2, 1, 7 },
   {1, 5, 2, 2, 1 }};
```

```
; eax = index
 lea eax, [eax + 4 * eax] ; 5 * index
 lea eax, [pgh + 4 * eax] ; pgh + (20 * index)
```

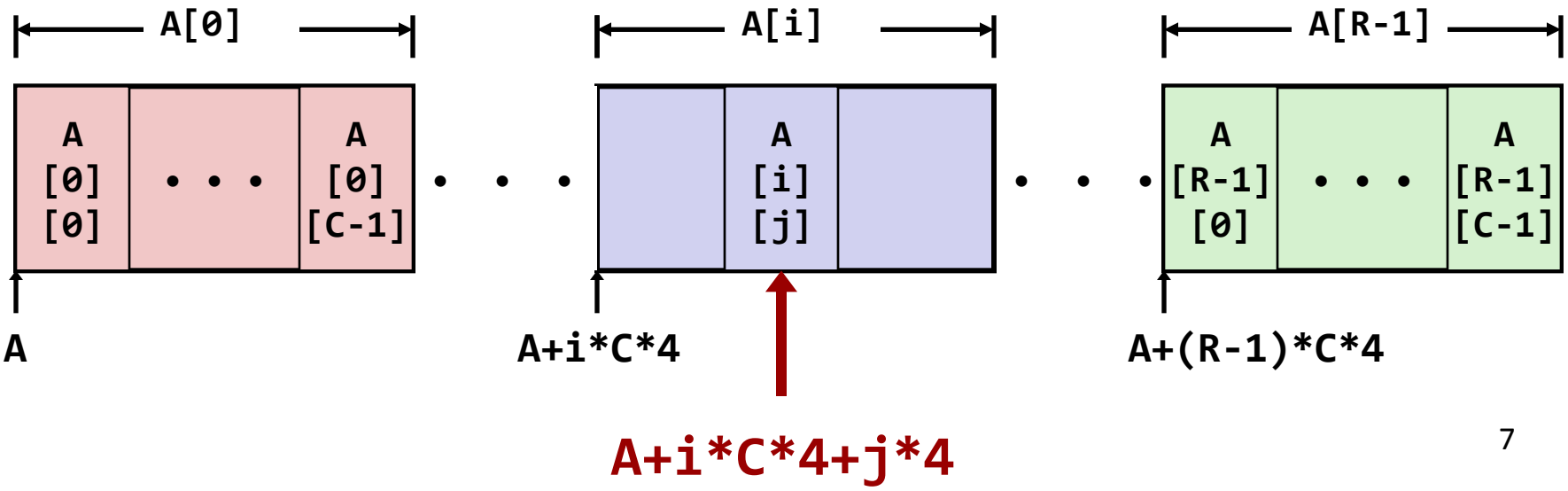
- **pgh[index]** массив из 5 **int**'ов
- Начальный адрес **pgh+20*index**

- Вычисляется и возвращается адрес
- Вычисление адреса в виде **pgh + 4*(index+4*index)**

© 2016 МГУ/ВМК/СП

- 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];
```



© 2016 МГУ/ВМК/СП

```
int get_pgh_digit (int index, int dig) {
  return pgh[index][dig];
}
```

```
mov     eax, dword [ebp + 8]          ; index
lea     eax, [eax + 4 * eax]          ; 5*index
add     eax, dword [ebp + 12]         ; 5*index+dig
mov     eax, dword [pgh + 4 * eax]    ; смещение 4*(5*index+dig)
```

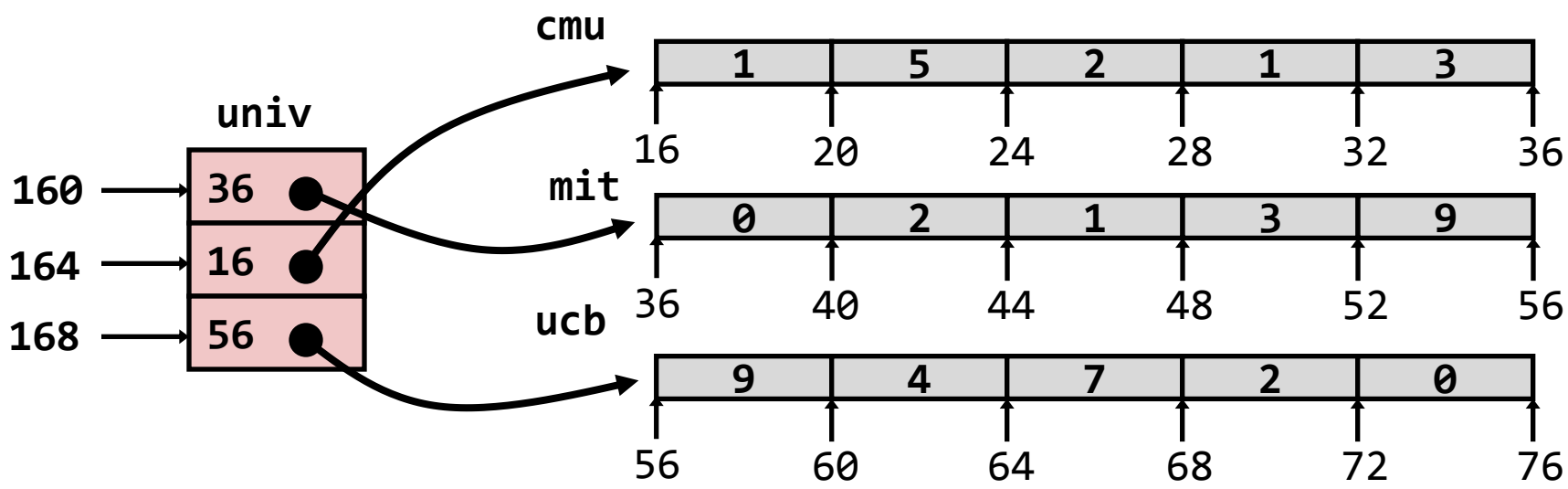
- **pgh[index][dig]** – тип **int**
- Адрес: **pgh + 20*index + 4*dig =**
  = pgh + 4*(5*index + dig)
- Вычисление адреса производится как
  pgh + 4*((index+4*index)+dig)

© 2016 МГУ/ВМК/СП

```
zip_dig cmu = { 1, 5, 2, 1, 3 };
zip_dig mit = { 0, 2, 1, 3, 9 };
zip_dig ucb = { 9, 4, 7, 2, 0 };
```

```
#define UCOUNT 3
int *univ[UCOUNT] = {mit, cmu, ucb};
```

- Переменная **univ** представляет собой массив из 3 элементов
- Каждый элемент – указатель (размером 4 байта)
- Каждый указатель ссылается на массив из **int**'ов



© 2016 МГУ/ВМК/СП

```
int get_univ_digit (int index, int dig) {
  return univ[index][dig];
}
```

```
mov    eax, dword [ebp + 8]              ; index
mov    edx, dword [univ + 4 * eax]       ; p = univ[index]
mov    eax, dword [ebp + 12]             ; dig
mov    eax, dword [edx + 4 * eax]        ; p[dig]
```

- Доступ к элементу **`Mem[Mem[univ+4*index]+4*dig]`**
- Необходимо выполнить два чтения из памяти
  - Первое чтение получает указатель на одномерный массив
  - Затем второе чтение выполняет выборку требуемого элемента этого одномерного массива

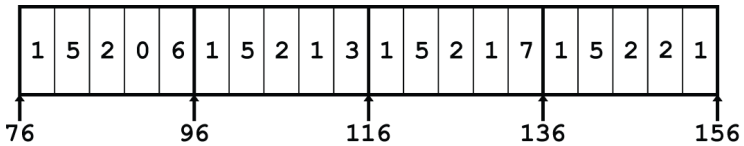
© 2016 МГУ/ВМК/СП

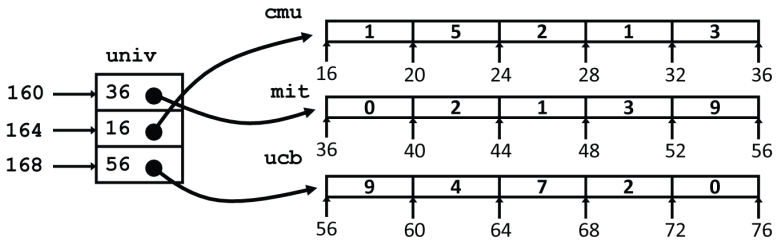
Многомерный массив

```
int get_pgh_digit
  (int index, int dig)
{
  return pgh[index][dig];
}
```

Многоуровневый массив

```
int get_univ_digit
  (int index, int dig)
{
  return univ[index][dig];
}
```





- Значительное внешнее сходство в Си
- Существенное различие в ассемблере

`Mem[pgh+20*index+4*dig]`

`Mem[Mem[univ+4*index]+4*dig]`

© 2016 МГУ/ВМК/СП

Матрица N X N

- Фиксированные размерности
  - Значение N известно во время компиляции

```
#define N 16
typedef int fix_matrix[N][N];
/* Get element a[i][j] */
int fix_ele
  (fix_matrix a, int i, int j)
{
  return a[i][j];
}
```

- Динамически задаваемая размерность. Требуется явное преобразование индексов
  - Традиционный способ реализации динамических массивов

```
#define IDX(n, i, j) ((i)*(n)+(j))
/* Get element a[i][j] */
int vec_ele
 (int n, int *a, int i, int j)
{
  return a[IDX(n,i,j)];
}
```

- Динамически задаваемая размерность с неявной индексацией.
  - Поддерживается последними версиями gcc

```
/* Get element a[i][j] */
int var_ele
 (int n, int a[n][n], int i, int j) {
  return a[i][j];
}
```

© 2016 МГУ/ВМК/СП

Матрица 16 X 16

- **Доступ к элементу матрицы**
  - Адрес **A** + $i * (C * K) + j * K$
  - C = 16, K = 4

```
/* Получение элемента a[i][j] */
int fix_ele(fix_matrix a, int i, int j) {
  return a[i][j];
}
```

```
mov     edx, dword [ebp + 12]      ; i
sal     edx, 6                     ; i*64
mov     eax, dword [ebp + 16]      ; j
sal     eax, 2                     ; j*4
add     eax, dword [ebp + 8]       ; a + j*4
mov     eax, dword [eax + edx]     ; *(a + j*4 + i*64)
```

© 2016 МГУ/ВМК/СП

Матрица n X n

- **Доступ к элементу матрицы**
  - Адрес **A** + $i * (C * K) + j * K$
  - C = n, K = 4

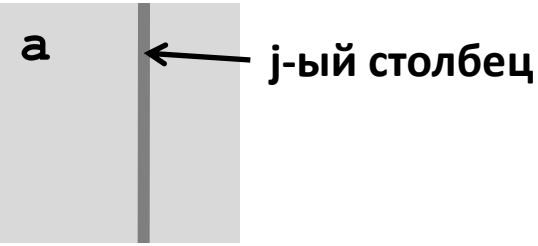
```
sizeof(a) = ?
sizeof(a[i]) = ?
```

```
/* Получение элемента  a[i][j] */
int var_ele(int n, int a[n][n], int i, int j) {
  return a[i][j];
}
```

```
mov     edx, dword [ebp + 8]        ; n
sal     edx, 2                      ; n*4
imul    edx, dword [ebp + 16]       ; i*n*4
mov     eax, dword [ebp + 20]       ; j
sal     eax, 2                      ; j*4
add     eax, dword [ebp + 12]       ; a + j*4
mov     eax, dword [eax + edx]      ; *(a + j*4 + i*n*4)
```

© 2016 МГУ/ВМК/СП

Оптимизация доступа к элементам массива




```
#define N 16
typedef int fix_matrix[N][N];
```

- Вычисления
  - Проход по всем элементам в столбце j
- Оптимизация
  - Выборка последовательных элементов из отдельного столбца

```
/* Выборка столбца j из массива */
void fix_column
  (fix_matrix a, int j, int *dest)
{
  int i;
  for (i = 0; i < N; i++)
    dest[i] = a[i][j];
}
```

© 2016 МГУ/ВМК/СП

Оптимизация доступа к элементам массива

- Оптимизация
  - Вычисляем `ajp = &a[i][j]`
    - Начальное значение `a + 4*j`
    - Шаг `4*N`

Регистр	Значение
ecx	ajp
ebx	dest
edx	i

```
/* Выборка столбца j из массива */
void fix_column
  (fix_matrix a, int j, int *dest)
{
  int i;
  for (i = 0; i < N; i++)
    dest[i] = a[i][j];
}
```

```
.L8:                                          ; loop:
   mov    eax, dword [ecx]                    ; считываем *ajp
   mov    dword [ebx + 4 * edx], eax          ; сохраняем в dest[i]
   add    edx, 1                              ; i++
   add    ecx, 64                             ; ajp += 4*N
   cmp    edx, 16                             ; i vs. N
   jne    .L8                                 ; if !=, goto loop
```

© 2016 МГУ/ВМК/СП

Оптимизация доступа к элементам массива

- Вычисляем `ajp = &a[i][j]`
  - Начальное значение `a + 4*j`
  - Шаг `4*n`

Регистр	Значение
ecx	ajp
edi	dest
edx	i
ebx	4*n
esi	n

```
/* Выборка столбца j из массива */
void var_column
  (int n, int a[n][n],
   int j, int *dest)
{
  int i;
  for (i = 0; i < n; i++)
    dest[i] = a[i][j];
}
```

```
.L18:                                          ; loop:
   mov    eax, dword [ecx]                     ; считываем *ajp
   mov    dword [edi + 4 * edx], eax           ; сохраняем в dest[i]
   add    edx, 1                               ; i++
   add    ecx, ebx                             ; ajp += 4*n
   cmp    esi, edx                             ; n vs. i
   jg     .L18                                 ; if (>) goto loop
```

© 2016 МГУ/ВМК/СП

Оптимизация доступа к элементам массива

- Изменение направления прохода по циклу
  - Выход из цикла по нулевому счетчику
  - Шаг отрицательный
  - Меняются начальные значения указателей
  - Достаточно вывести к нулю один из индексов

```
/* Выборка столбца j из массива */
void var_column
  (int n, int a[n][n],
   int j, int *dest) {

  int i;
  for (i = n-1; i >=0; i--) {
    dest[i] = a[i][j];
  }
}
```

```
.L18:                                          ; loop:
   mov    eax, dword [ecx]                      ; считываем *ajp
   mov    dword [edi + 4 * edx], eax            ; сохраняем в dest[i]
   add    edx, 1                                ; i++
   add    ecx, ebx                              ; ajp += 4*n
   cmp    esi, edx                              ; n vs. i
   jg     .L18                                  ; if (>) goto loop
```

© 2016 МГУ/ВМК/СП

Оптимизация доступа к элементам массива

Регистр	Начальное значение
ecx	a+4*n*(n-1)+4*j
edi	dest – 4
edx	n
ebx	4*n
esi	освободился

```
/* Выборка столбца j из массива */
void var_column
  (int n, int a[n][n],
   int j, int *dest) {

  int i;
  dest--;
  for (i = n; i != 0; i--)
    dest[i] = a[i-1][j];
}
```

```
.L18:                                          ; loop:
   mov    eax, dword [ecx]                     ; считываем *(ajp+…)
   mov    dword [edi + 4 * edx], eax           ; сохраняем в dest[i]
   sub    ecx, ebx                             ; ajp -= 4*n
   sub    edx, 1                               ; i--
   jnz    .L18                                 ; if (!=) goto loop
```

© 2016 МГУ/ВМК/СП

Обратная задача

```
; пролог функции пропущен
mov  ecx, dword [ebp + 8]       ; 1
mov  edx, dword [ebp + 12]      ; 2
lea  eax, [8 * ecx]             ; 3
sub  eax, ecx                   ; 4
add  eax, edx                   ; 5
lea  edx, [edx + 4 * edx]       ; 6
add  edx, ecx                   ; 7
mov  eax, dword [m1 + 4 * eax]  ; 8
add  eax, dword [m2 + 4 * edx]  ; 9
; эпилог функции пропущен
```

M = ?, N = ?

```
int m1[M][N];
int m2[N][M];

int sum_element(int i, int j) {
   return m1[i][j] + m2[j][i];
}
```

© 2016 МГУ/ВМК/СП

Типы данных языка Си
Далее

- `char`
- Стандартные знаковые целочисленные типы
  - `signed char`
  - `short int`
  - `int`
  - `long int`
  - `long long int`
- Стандартные беззнаковые целочисленные типы
  - `_Bool`
- Перечисление
- Типы чисел с плавающей точкой
  - `float`
  - `double`
  - `long double`
  - `_Complex`
- Производные типы
  - Массивы
  - **Структуры**
  - **Объединения**
  - Указатели
  - Указатели на функции

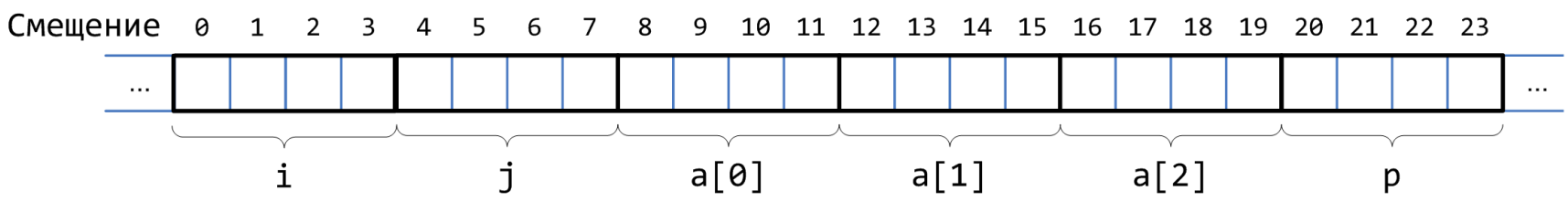
© 2016 МГУ/ВМК/СП

Структуры

```
struct rec {
   int i;
   int j;
   int a[3];
   struct rec *p;
}
```

- Непрерывный блок памяти
- Обращение к полям структуры осуществляется по их именам
- Поля могут быть разных типов

Расположение в памяти

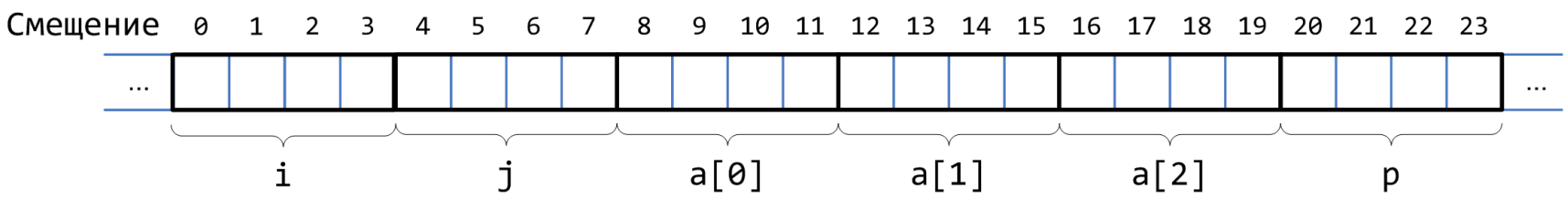


© 2016 МГУ/ВМК/СП

Доступ к полям

```
struct rec {
   int i;
   int j;
   int a[3];
   struct rec *p;
}
```

- `struct rec *x` – указатель на первый байт структуры
- Каждое поле расположено на определенном смещении от начала структуры




```
static struct rec *x;
...
x->j = x->i;
```

```
mov edx, dword [x]        ; (1)
mov eax, dword [edx]      ; (2)
mov dword [edx + 4], eax  ; (3)
```

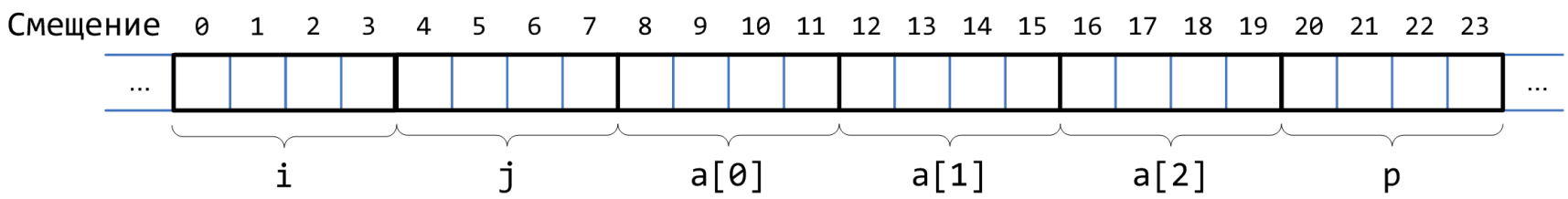
© 2016 МГУ/ВМК/СП

Указатель на поле структуры

```
struct rec {
   int i;
   int j;
   int a[3];
   struct rec *p;
}
```

- Смещение каждого поля известно во время компиляции

```
static struct rec *x;
static int i;
...
&(x->a[i]);
```




```
mov edx, dword [i]            ; (1)
mov eax, dword [x]            ; (2)
lea eax, [eax + 4 * edx + 8] ; (3)
```

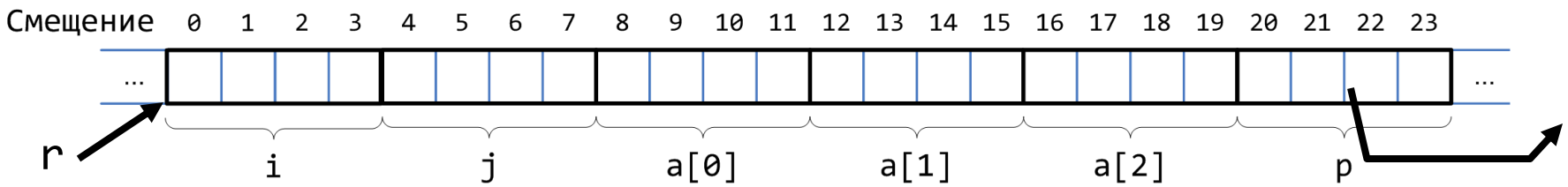
© 2016 МГУ/ВМК/СП

- Проход по связанному списку

```
void set_val (struct rec *r, int val) {
  while (r) {
    int i = r->i;
    r->a[i] = val;
    r = r->p;
  }
}
```

```
struct rec {
   int i;
   int j;
   int a[3];
   struct rec *p;
}
```

Регистр	Переменная
edx	r
ecx	val



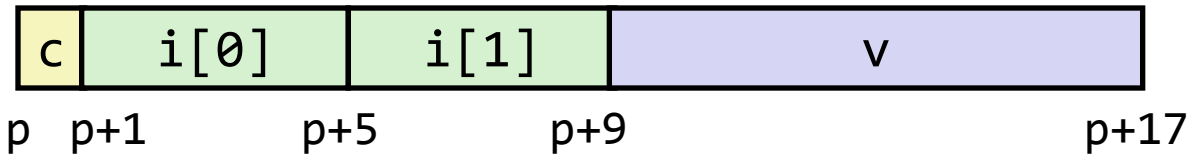

```
.L17: ; цикл
   mov  eax, [edx]                  ; r->i
   mov  [edx + 4 * eax + 8], ecx ; r->a[i] = val
   mov  edx, [edx + 20]             ; r = r->p
   test edx, edx                    ; r?
   jne  .L17                        ; If != 0 goto .L17
```

© 2016 МГУ/ВМК/СП

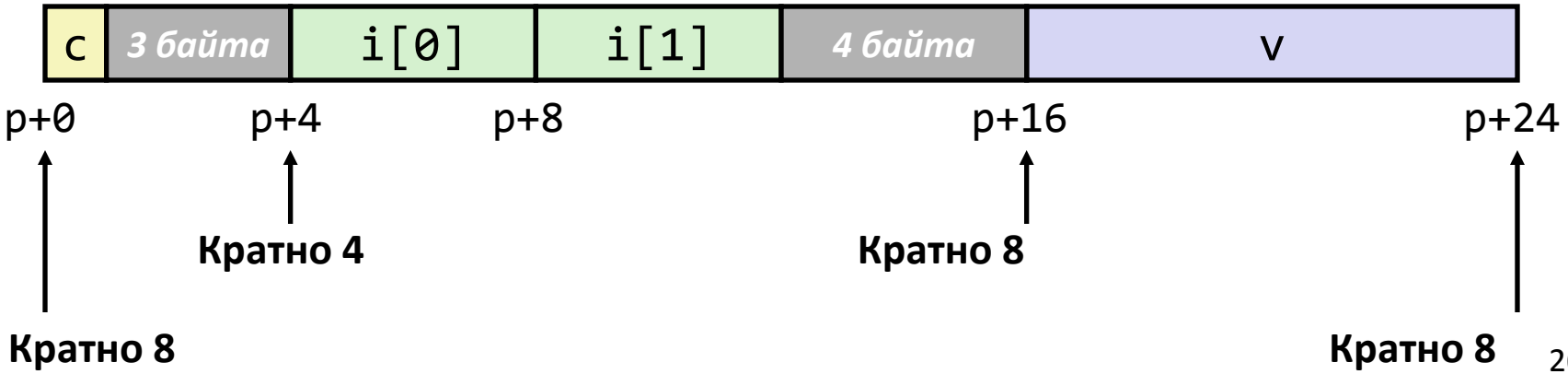
Выравнивание полей в структурах

```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```

- Невыровненные данные



- Выровненные данные
  - Если примитивный тип данных требует ***K*** байт
  - Адрес должен быть кратен ***K***



© 2016 МГУ/ВМК/СП

Почему выравнивают данные

- Выровненные данные
  - Размер примитивного типа данных ***K*** байт
  - Адрес должен быть кратен ***K***
  - Для некоторых архитектур это требование обязательно должно выполняться
  - Для IA-32 требование к выравниванию имеет рекомендательный характер
    - Требования различаются для IA-32/x86-64, Linux/Windows
- Причины
  - Доступ к физической памяти осуществляется блоками (выровненными) по 4 или 8 байт (зависит от аппаратуры)
    - Эффективность теряется при обращении к данным, расположенным в двух блоках
    - Виртуальная память…
- Компилятор
  - Расставляет пропуски между полями для сохранения выравнивания

© 2016 МГУ/ВМК/СП

Правила выравнивания (IA-32)

- 1 байт : char, …
  - Ограничений нет
- 2 байта : short, …
  - Младший бит адреса должен быть 0_2
- 4 байта : int, long, float, char *, …
  - Два младших бита адреса должны быть 00_2
- 8 байт : double, …
  - Windows (и другие …):
    - Младшие три бита адреса должны быть 000_2
  - Linux:
    - Два младших бита адреса должны быть 00_2
    - Т.е. рассматриваются как и 4-байтные примитивные типы данных
- 12 байт : long double (gcc)
  - Windows, Linux:
    - Два младших бита должны быть 00_2
    - Т.е. рассматриваются как и 4-байтные примитивные типы данных

© 2016 МГУ/ВМК/СП

Правила выравнивания (x86-64)

- 1 байт : char, …
  - Ограничений нет
- 2 байта : short, …
  - Младший бит адреса должен быть 0_2
- 4 байта : int, float, …
  - Два младших бита адреса должны быть 00_2
- 8 байт: double, long, char *, …
  - Windows & Linux:
    - Младшие три бита адреса должны быть 000_2
- 16 байт: long double
  - Linux:
    - Младшие три бита адреса должны быть 000_2
    - Т.е. рассматриваются как и 8-байтные примитивные типы данных

Тип long double в компиляторе MS VC
- 16-разрядная архитектура
  - sizeof(long double) = 10 // 80 бит
- 32-разрядная архитектура и далее …
  - long double ≡ double