Лекция 9

12 марта

Регистры и типы данных

- Целые числа
  - Размещаются и обрабатываются в регистрах общего назначения
  - Знаковые/беззнаковые числа

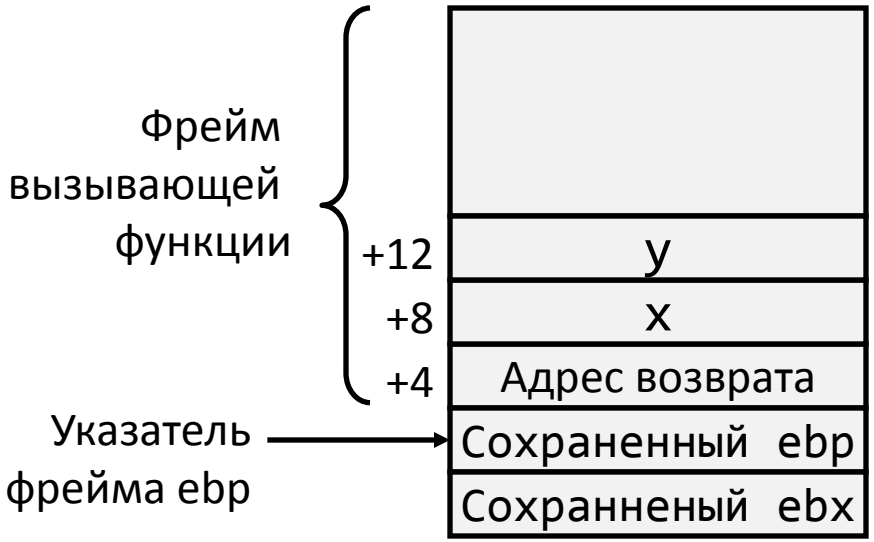
Intel	ASM	Bytes	C
byte	**b**	1	**[unsigned] char**
word	**w**	2	**[unsigned] short**
double word	**d**	4	**[unsigned] int**
quad word	**q**	8	**[unsigned] long long int**

- Указатели
- Числа с плавающей точкой
  - Размещаются и обрабатываются в специализированных регистрах для чисел с плавающей точкой

Intel	ASM	Bytes	C
Single	**d**	4	**float**
Double	**q**	8	**double**

Пример1: обмен значениями с использованием указателей

```
void exchange(int *x, int *y) {
   int tmp = *x;
   *x = *y;
   *y = tmp;
}
```




```
exchange:
   push ebp
   mov  ebp, esp
   push ebx
   mov  edx, dword [ebp+8]
   mov  ecx, dword [ebp+12]
   mov  ebx, dword [edx]
   mov  eax, dword [ecx]
   mov  dword [edx], eax
   mov  dword [ecx], ebx
   pop  ebx
   pop  ebp
   ret
```

Пример2: обмен значениями

Используем указатели без дополнительной переменной

```
; gcc -O2 -S -masm=intel xorcopy.c
...
exchange:
   push    ebp
   mov     ebp, esp
   mov     ecx, dword [ebp+12]
   push    ebx
   mov     ebx, dword [ebp+8]
   mov     eax, dword [ecx]
   mov     edx, eax
   xor     edx, dword [ebx]
   xor     eax, edx
   mov     dword [ecx], eax
   xor     eax, edx
   mov     dword [ebx], eax
   pop     ebx
   pop     ebp
   ret
...
```

```
#include <stdio.h>

void exchange(int *x, int *y) {
   *x ^= (*y ^= (*x ^= *y));
}

int main() {
   int x = 0, y = 1;
   printf("Before x = %d, y = %d\n", x, y);
   exchange(&x, &y);
   printf("After  x = %d, y = %d\n", x, y);
   return 0;
}
```

```
-bash-2.05b$ gcc -O2 -o xorcopy xorcopy.c
-bash-2.05b$ ./xorcopy
Before x = 0, y = 1
After  x = 1, y = 0
-bash-2.05b$ gcc -O0 -o xorcopy xorcopy.c
-bash-2.05b$ ./xorcopy
Before x = 0, y = 1
After  x = 0, y = 0
```

Результат изменился!!!

```
; gcc -O0 -S -masm=intel xorcopy.c
...
   mov     eax, dword [ebp+8]
   mov     ebx, dword [eax]
   mov     eax, dword [ebp+12]
   mov     ecx, dword [eax]
   mov     eax, dword [ebp+8]
   mov     edx, dword [eax]
   mov     eax, dword [ebp+12]
   mov     eax, dword [eax]
   xor     edx, eax
   mov     eax, dword [ebp+8]
   mov     dword [eax], edx
   mov     eax, dword [ebp+8]
   mov     eax, dword [eax]
   mov     edx, ecx
   xor     edx, eax
   mov     eax, dword [ebp+12]
   mov     dword [eax], edx
   mov     eax, dword [ebp+12]
   mov     eax, dword [eax]
   mov     edx, ebx
   xor     edx, eax
; в xor используется *x (edx = 0),
; вычисленный до первого присваивания
   mov     eax, dword [ebp+8]
   mov     dword [eax], edx
...
```

```
#include <stdio.h>

void exchange(int *x, int *y) {
   *x ^= (*y ^= (*x ^= *y));
}

int main() {
   int x = 0, y = 1;
   printf("Before x = %d, y = %d\n", x, y);
   exchange(&x, &y);
   printf("After  x = %d, y = %d\n", x, y);
   return 0;
}
```

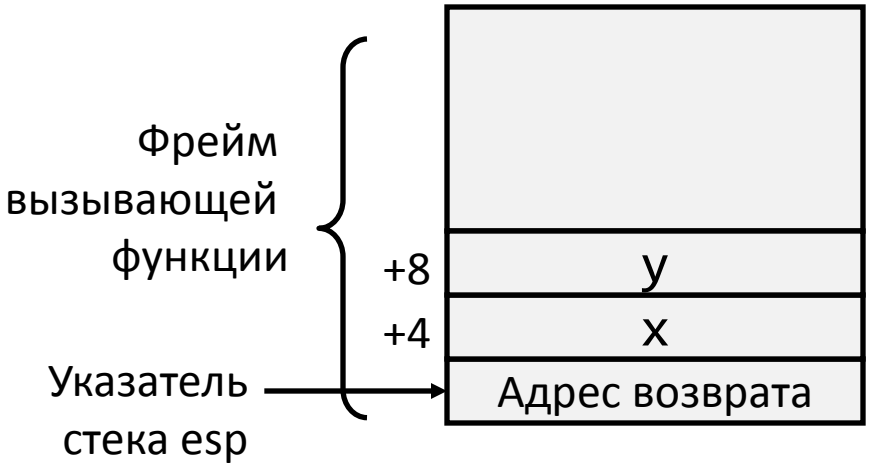
```
-bash-2.05b$ gcc -O2 -o xorcopy xorcopy.c
-bash-2.05b$ ./xorcopy
Before x = 0, y = 1
After  x = 1, y = 0
-bash-2.05b$ gcc -O0 -o xorcopy xorcopy.c
-bash-2.05b$ ./xorcopy
Before x = 0, y = 1
After  x = 0, y = 0
```

Пример3: правильный обмен с использованием команд XOR

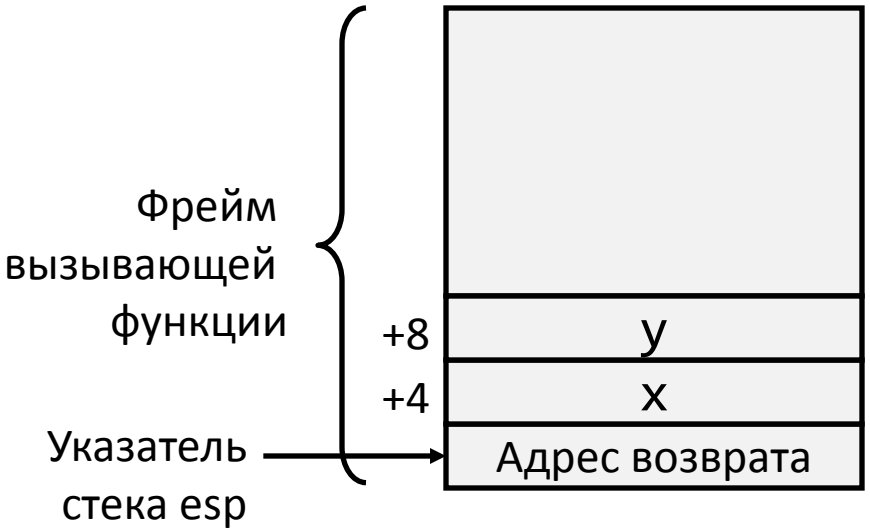
```
void exchange(int *x, int *y) {
   if (x == y) {
      return;
   }

   *x ^= *y;
   *y ^= *x;
   *x ^= *y;
}
```

Фрейм намеренно не создается

```
exchange:
   mov     ecx, dword [esp+4]
   mov     edx, dword [esp+8]
   cmp     ecx, edx
   je      .L3
   mov     eax, dword [edx]
   xor     eax, dword [ecx]
   mov     dword [ecx], eax
   xor     eax, dword [edx]
   mov     dword [edx], eax
   xor     dword [ecx], eax
.L3:
   ret
```



Пример4: обмен значениями с использованием указателей




```
void exchange(int *x, int *y) {
   int tmp = *x;
   *x = *y;
   *y = tmp;
}
```

Регистр	Значение
edx	**tmp**

Фрейм намеренно не создается

```
exchange:
   mov eax, [esp+4] ; eax ← x
   mov edx, [eax]   ; edx ← *x
   mov ecx, [esp+8] ; ecx ← y
   mov ecx, [ecx]   ; ecx ← *y
   mov [eax], ecx   ; *x  ← ecx
   mov eax, [esp+8] ; eax ← y
   mov [eax], edx   ; *y  ← edx
```

```
int tmp = *x;
```

```
*x = *y;
```

```
*y = tmp;
```

Обратная задача

```
void f(int *xp, int *yp, int *zp) {
        ???
}
```

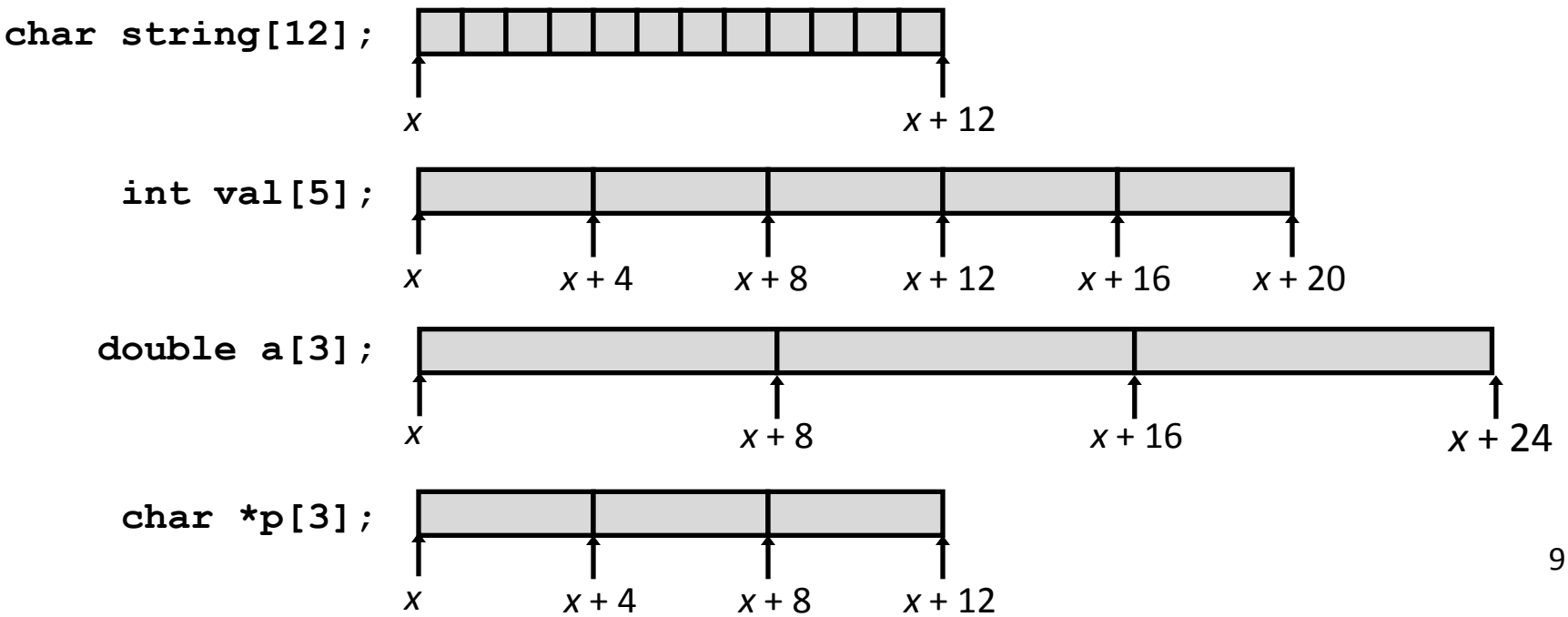
```
exchange:
   ... ; пролог функции
   mov edi, [ebp + 8]   ; (1)
   mov edx, [ebp + 12]  ; (2)
   mov ecx, [ebp + 16]  ; (3)
   mov ebx, [edx]       ; (4)
   mov esi, [ecx]       ; (5)
   mov eax, [edi]       ; (6)
   mov [edx], eax       ; (7)
   mov [ecx], ebx       ; (8)
   mov [edi], esi       ; (9)
   ... ; эпилог функции
```

Параметр	Размещение
xp	**ebp + 8**
yp	**ebp + 12**
zp	**ebp + 16**

Массивы – размещение в памяти

`T A[L];`

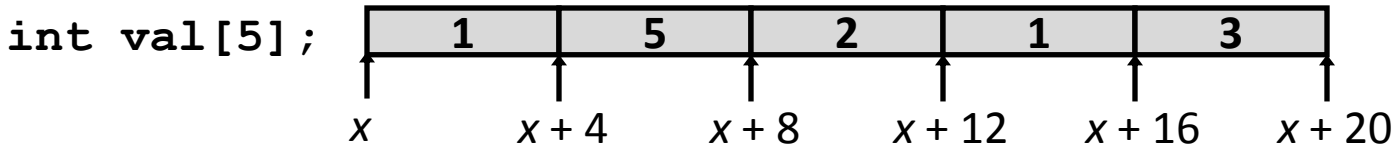
- Массив элементов типа T, размер массива – L
- Массив располагается в непрерывном блоке памяти размером *L* * **`sizeof`** (*T*) байт



Доступ к элементам массива

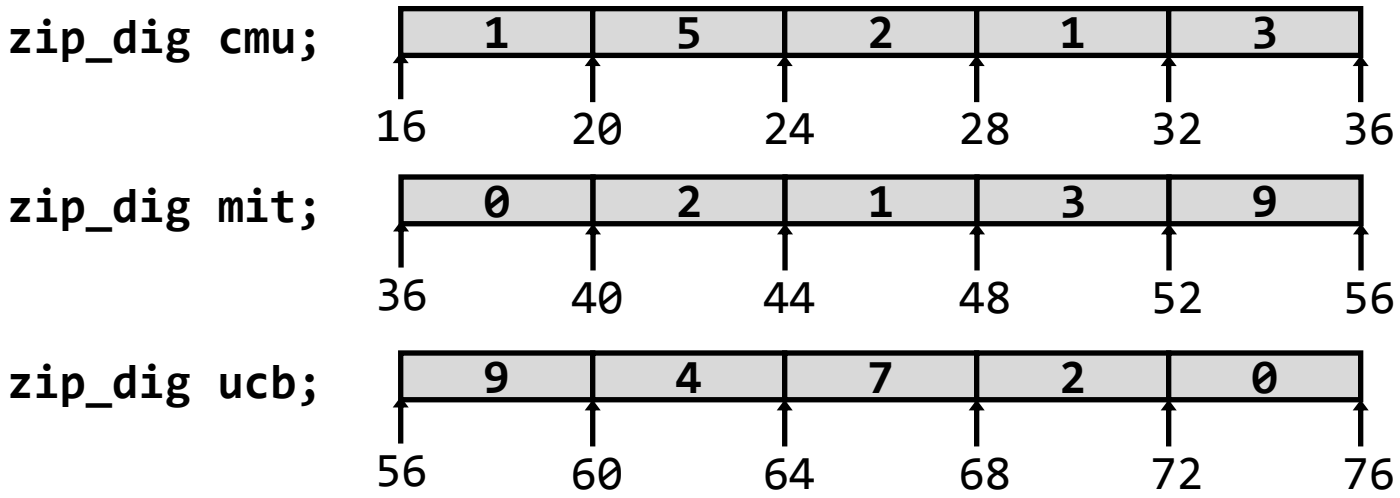
`T  A[L];`

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



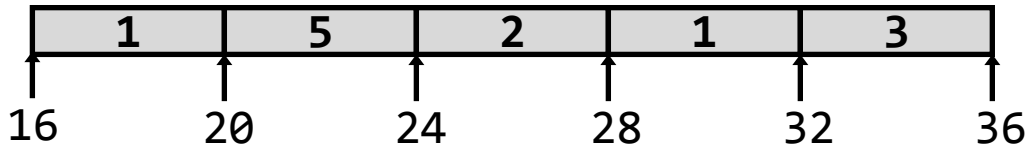
- Задачи …

```
#define ZLEN 5
typedef int zip_dig[ZLEN];

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



- Объявление переменной “zip_dig cmu” эквивалентно “int cmu[5]”
- Массивы были размещены в последовательно идущих блоках памяти размером 20 байт каждый
  - В общем случае не гарантируется, что массивы будут размещены непрерывно

```
zip_dig cmu;
```




```
int get_digit (zip_dig z, int dig) {
   return z[dig];
}
```

```
; edx = z
; eax = dig
mov eax, dword [edx+4*eax] ; z[dig]
```

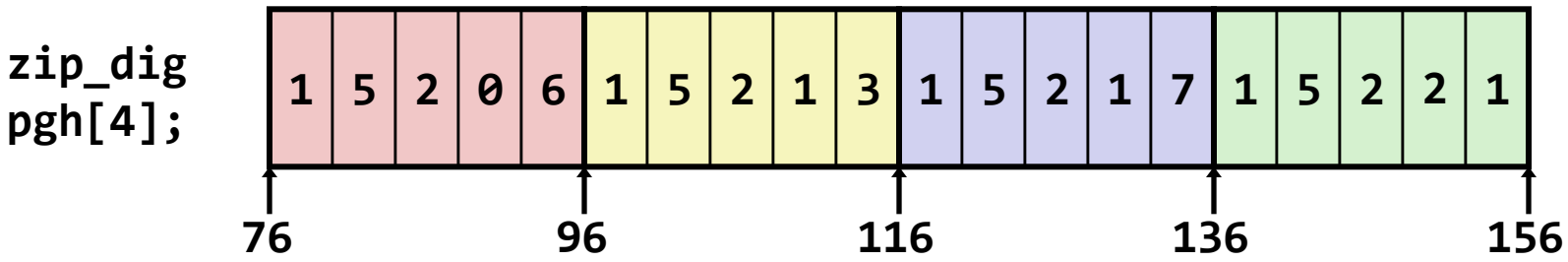
- Регистр edx содержит начальный (базовый) адрес массива
- Регистр eax содержит индекс элемента в массиве
- Адрес элемента edx + 4 * eax

```
void zincr(zip_dig z) {
  int i;
  for (i = 0; i < ZLEN; i++)
    z[i]++;
}
```

```
void zincr_v(zip_dig z) {
  void *vz = z;
  int i = 0;
  do {
    (*((int *) (vz + i))) += 1;
    i += ISIZE;
  } while (i != ISIZE * ZLEN);
}
```

```
                                  ;   edx = z
   mov eax, 0                     ;   eax = i
.L4:                              ; loop:
   add dword [edx + 4 * eax], 1 ;   z[i]++
   add eax, 1                     ;   i++
   cmp eax, 5                     ;   i vs. 5
   jne .L4                        ;   if (!=) goto loop
```

```
#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)

- Объявление

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

  – 2D массив элементов типа *T*

  – *R* строк, *C* столбцов

  – Размер типа *T* – *K* байт

- Размер массива

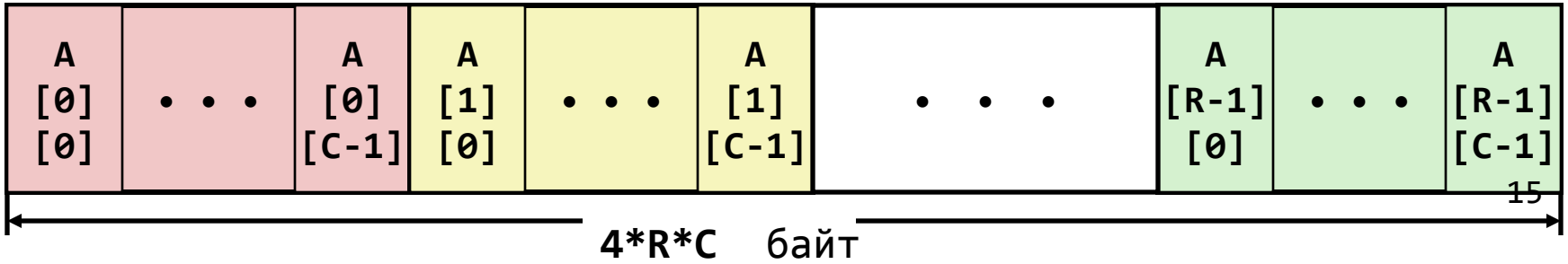
  – R * C * K байт

- Размещение в памяти

  – Развертывание по строкам

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

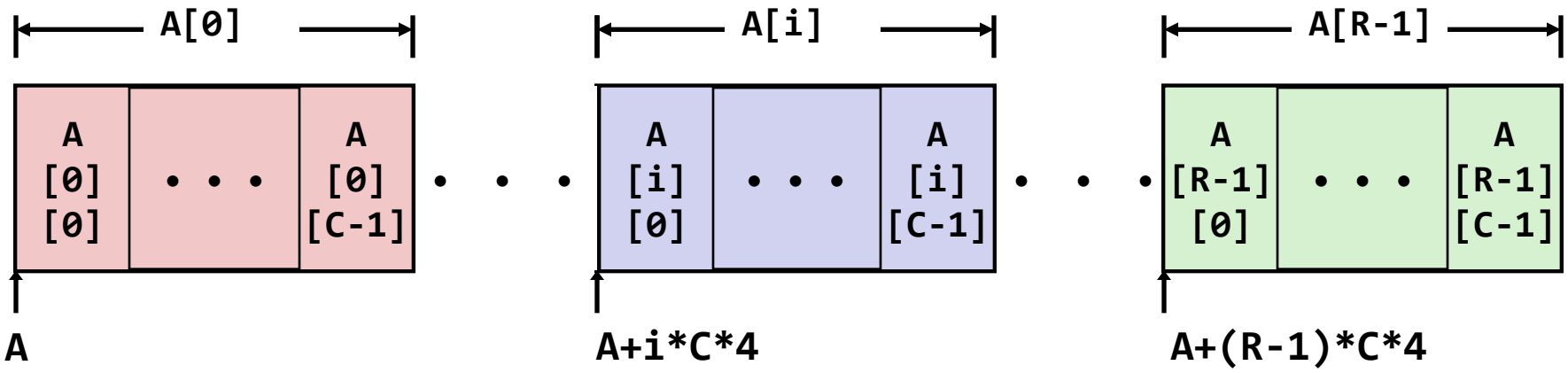
```
int A[R][C];
```



- 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];
```



```
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 }};
```

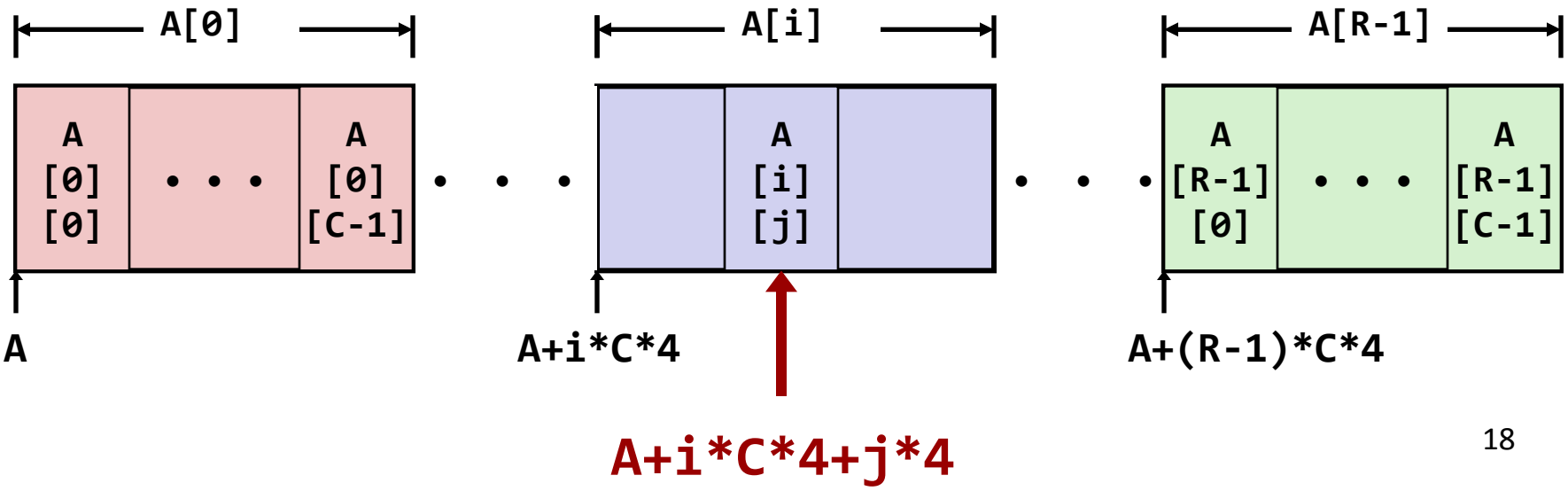
```
; eax = index
 lea eax, [eax + 4 * eax] ; 5 * index
 lea eax, [pgh + 4 * eax] ; pgh + (20 * index)
```

- **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)**

- 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];
```



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

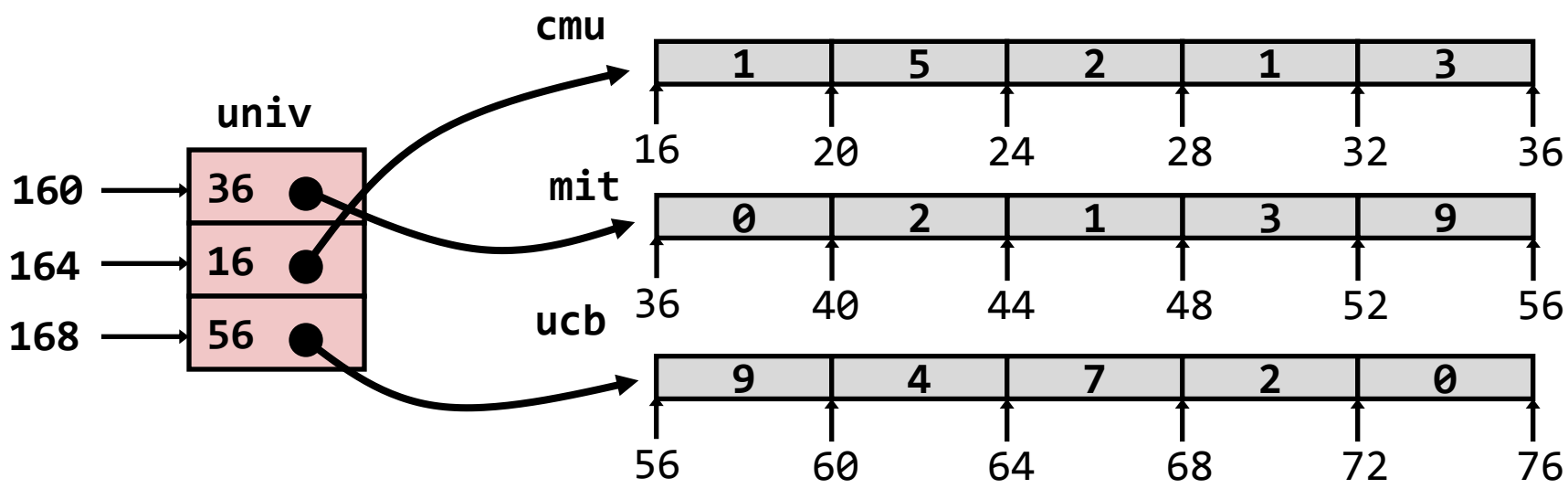
```
mov     eax, dword [ebp + 8]          ; index
lea     eax, [eax + 4 * eax]          ; 5*index
add     eax, dword [ebp + 12]         ; 5*index+dig
mov     eax, dword [pgh + 4 * eax]    ; смещение 4*(5*index+dig)
```

- **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)

```
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



```
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]
```

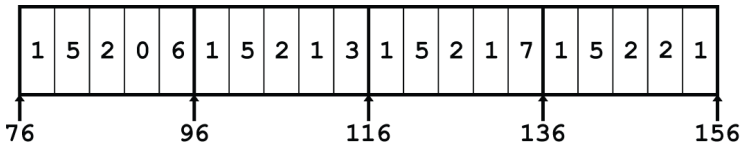
- Доступ к элементу **Mem[Mem[univ+4*index]+4*dig]**
- Необходимо выполнить два чтения из памяти
  - Первое чтение получает указатель на одномерный массив
  - Затем второе чтение выполняет выборку требуемого элемента этого одномерного массива

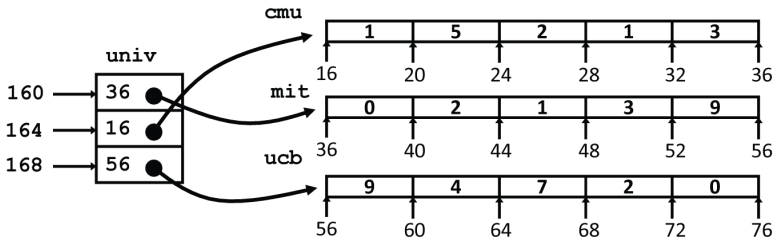
Многомерный массив

```
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]`