Лекция 9

10 марта

Обратная задача

```
int switchMeOnce(int x) {
   int result = 0;

   switch (x) {
   . . .
   }

   return result;
}
```

```
section .text
   . . .
   mov eax, dword [ebp-8]
   add eax, 2
   cmp eax, 6
   ja  .L2
   jmp [.L8 + 4*eax]
   . . .

section .rodata
  .L8 dd .L3, .L2, .L4, .L5, .L6, .L6, .L7
```

1. Сколько раз было использовано ключевое слово case?
2. Какие константы использовались?
3. Какие ветки выполнения были объединены?
4. Что помечено .L2?

Промежуточные итоги: передача управления

- Язык Си
  - if, if-else
  - do-while
  - while, for
  - switch
- Язык ассемблера
  - Условная передача управления
  - Условная передача данных
  - Косвенные переходы
- Стандартные приемы
  - Преобразования циклов к виду do-while
  - Использование таблицы переходов для операторов switch
  - Операторы switch с «разреженным» набором значений меток реализуются деревом решений
- Следующая тема: указатели и агрегатные типы данных

Типы данных языка Си

- `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

Регистры и типы данных

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

Intel	Асм.	байты	Си
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	Асм.	байты	Си
Single	d	4	float
Double	q	8	double

Обратная задача

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

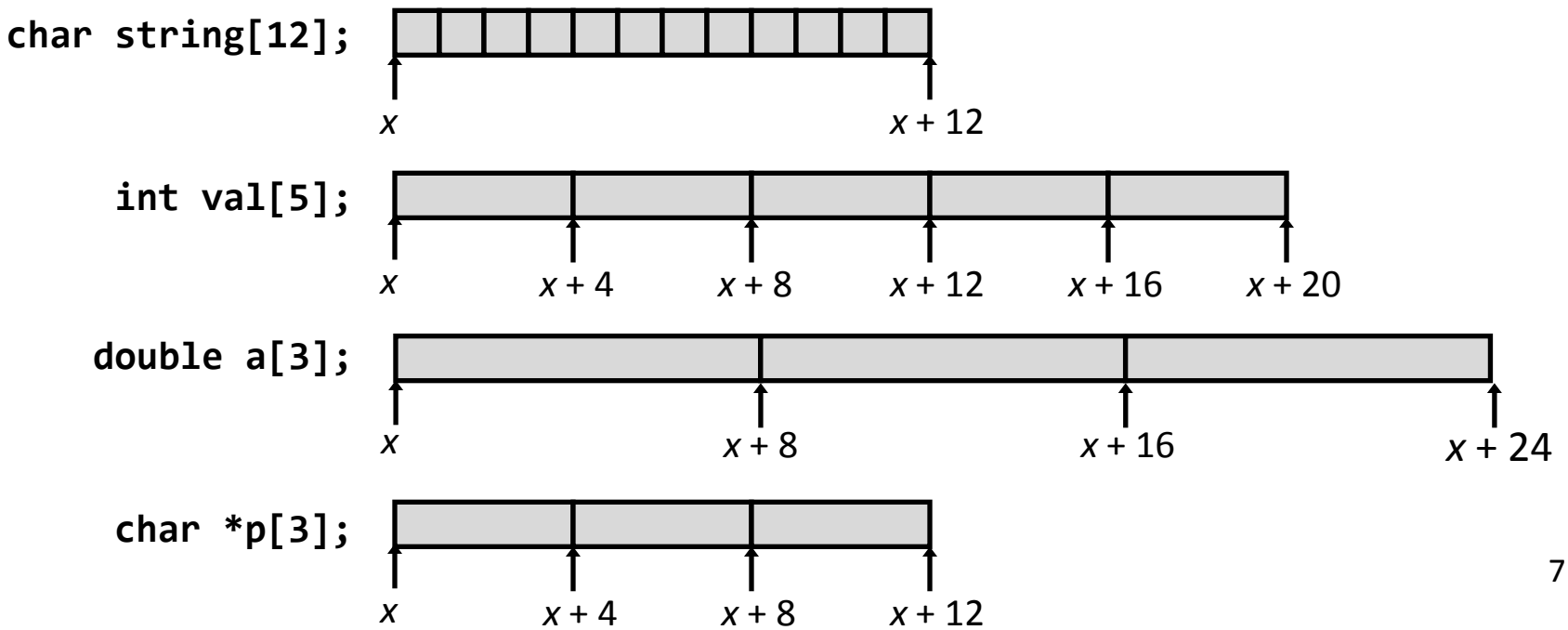
```
exchange:
   ... ; пролог функции
   mov edi, [ebp + 8]   ; (1)
   mov edx, [ebp + 12]  ; (2)
   mov ecx, [ebp + 16]  ; (3)
   mov ebx, [edx]       ; (4)
   mov esi, [ecx]       ; (5)
   mov eax, [edi]       ; (6)
   mov [edx], eax       ; (7)
   mov [ecx], ebx       ; (8)
   mov [edi], esi     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

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

Массивы – размещение в памяти

`T A[L];`

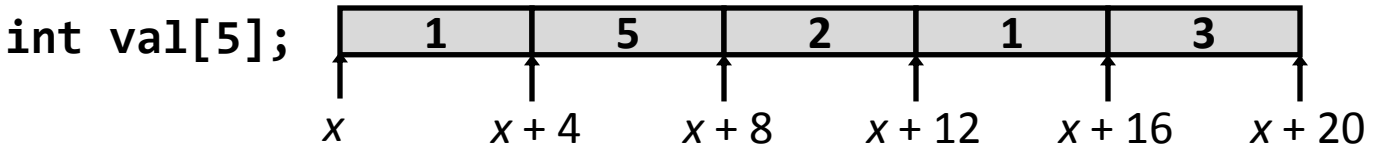
- Массив элементов типа **T**, размер массива – **L**
- Массив располагается в непрерывном блоке памяти размером **`L * sizeof(T)`** байт



Доступ к элементам массива

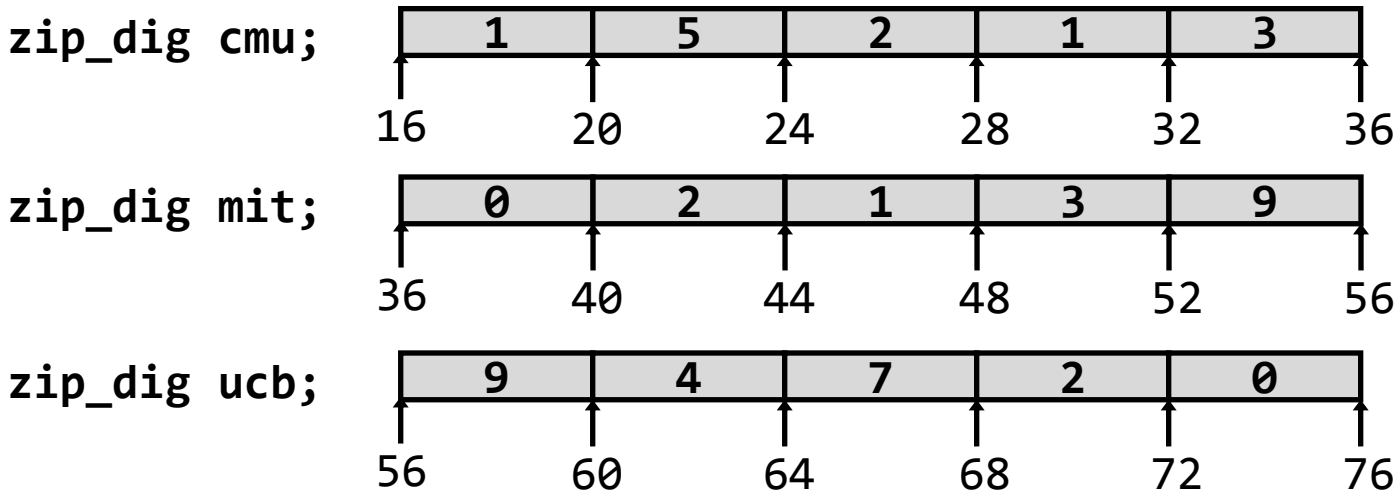
`T A[L];`

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

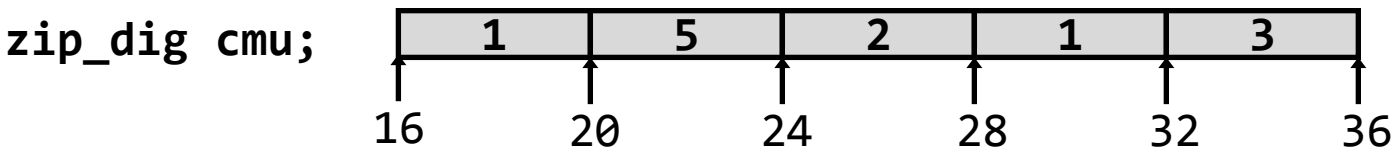


- Задачи …

```
#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




```
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

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

```c
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
```

Две обратные задачи

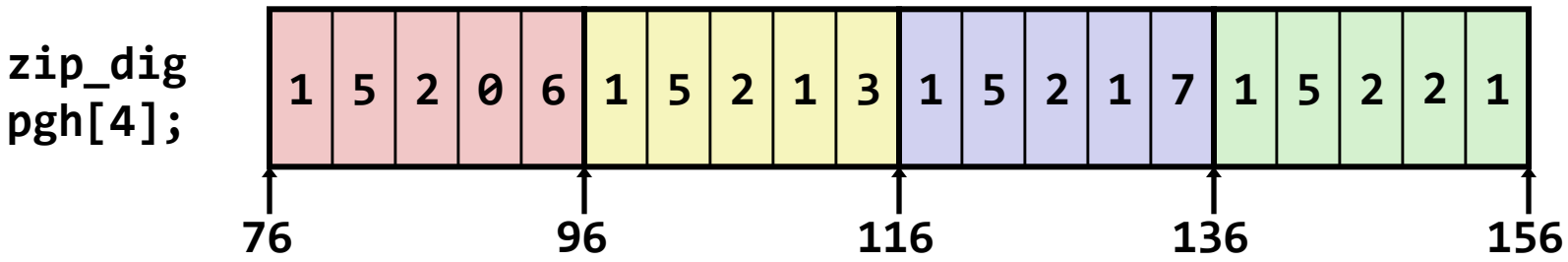
```
f:
   ...
   mov   edx, dword [ebp + 8]
   movsx ax, byte [a + edx]
   mov   word [b + edx + edx], ax
   ...
```

```
____________ 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) {
   _____________;
}
```

```
#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

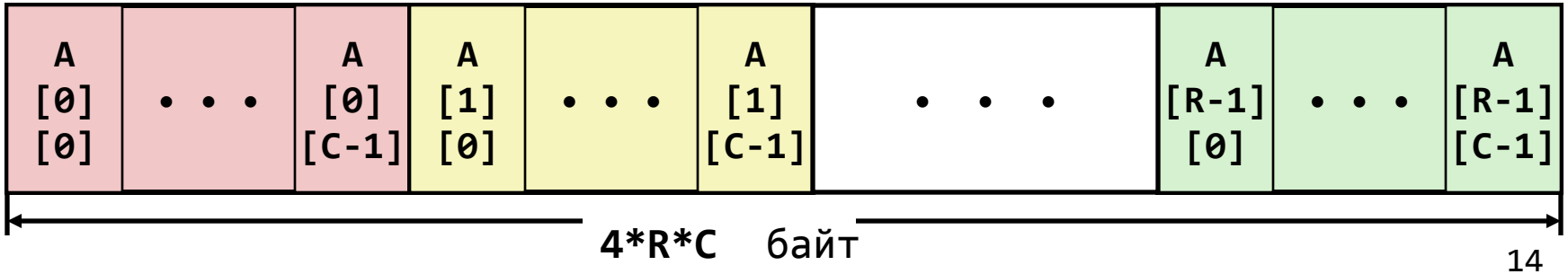
  – R * C * K байт

- Размещение в памяти

  – Развертывание по строкам

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

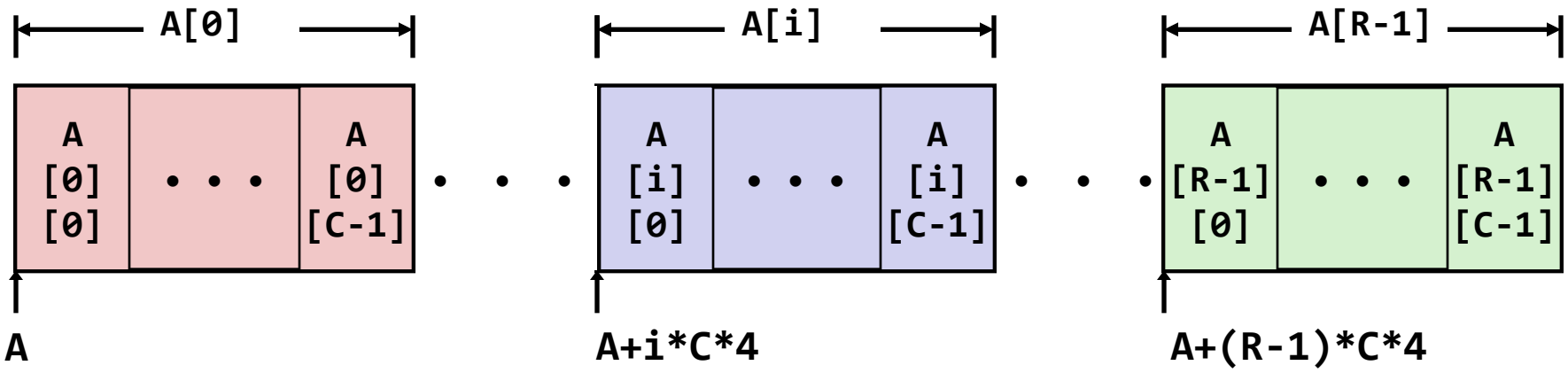
int A[R][C];



- 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 }};
```

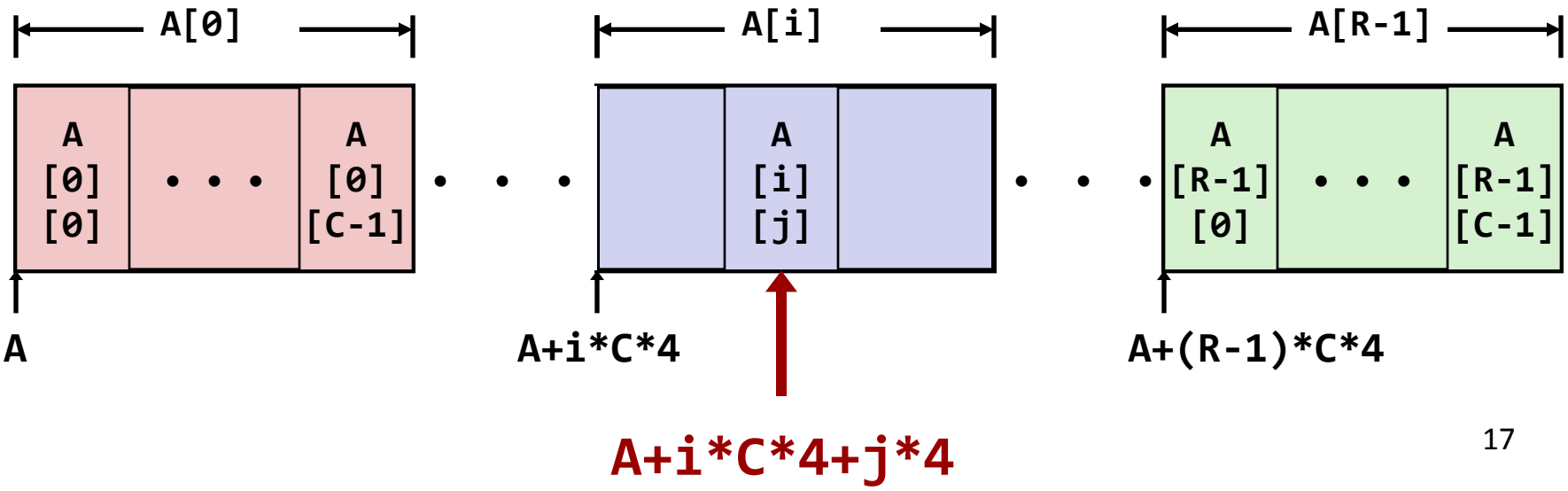
```
; eax = index
 lea eax, [eax + 4 * eax] ; 5 * index
 lea eax, [pgh + 4 * eax] ; pgh + (20 * index)
```

- **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)