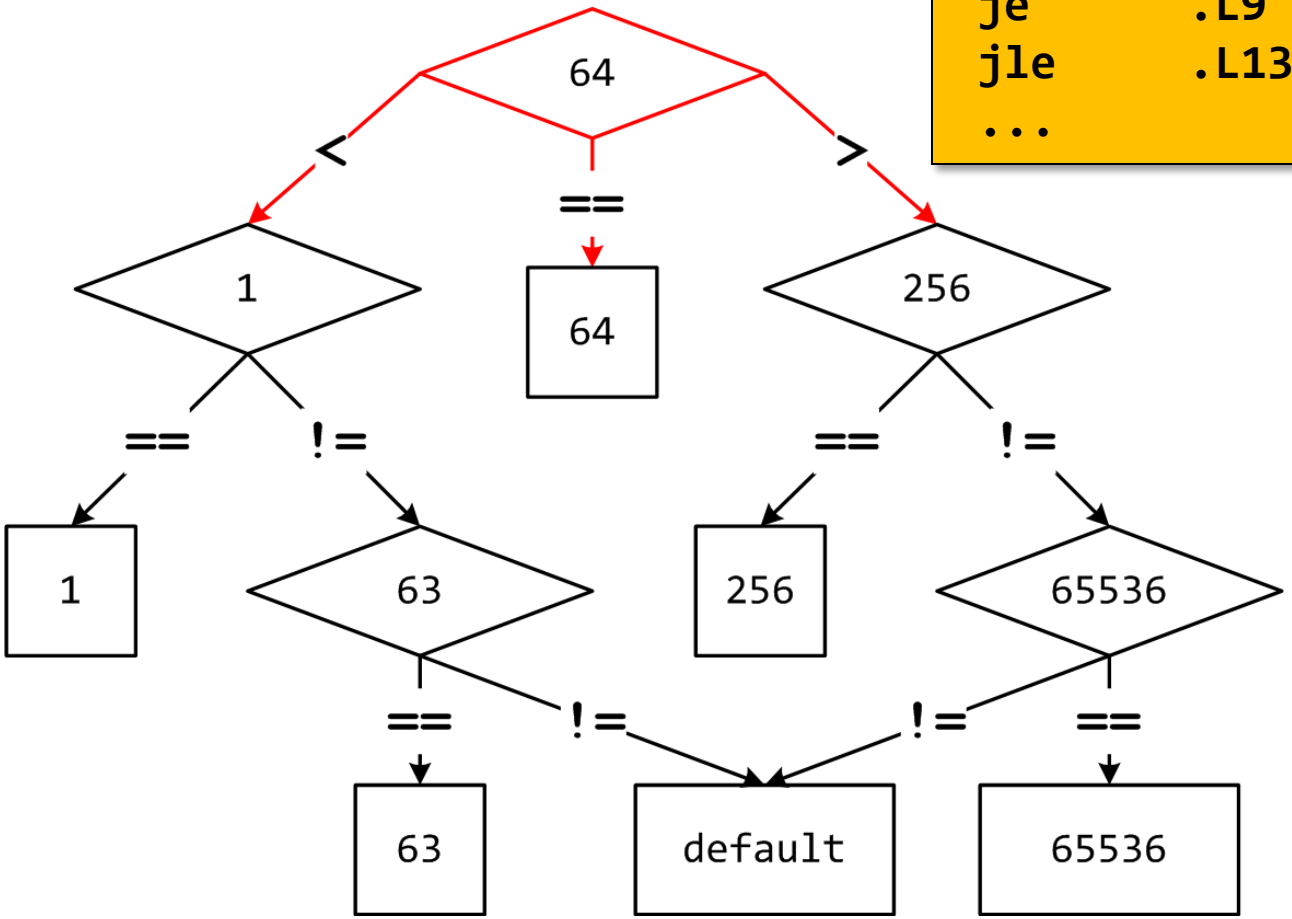
Лекция 9

07 марта

```
int f(int n, int *p) {
   int _res;
   switch (n) {
   default:
      _res = 0;
   /* «проваливаемся» */
   case 1:
      *p = _res;
      break;
   case 64:
      _res = 1;
      break;
   case 63:
      _res = 2;
      *p = _res;
    /* «проваливаемся» */
   case 256:
      _res = 3;
      break;
   case 65536:
      _res = 4;
   }
   return _res;
}
```

- Случай default расположен первым
- Управление «проваливается» в случаях default и 63
- Таблица переходов получается неприемлемо большой

```
...
push    ebp
mov     ebp, esp
mov     eax, 1
mov     edx, dword [ebp+8] ; n
cmp     edx, 64
je      .L9
jle     .L13
...
```



Обратная задача

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

   switch (x) {
   . . .
   }

   return result;
}
```

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

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

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

```
 .L7 dd .L3, .L2, .L4, .L2
     dd .L5, .L6, .L2, .L4
  . . .
     mov eax, dword [ebp+8]
     cmp eax, 7
     ja  .L2
     jmp [.L7 + 4*eax]
.L2: mov eax, dword [ebp+12]
     jmp .L8
.L5: mov eax, 4
     jmp .L8
.L6: mov eax, dword [ebp+12]
     xor eax, 15
     mov dword [ebp+16], eax
.L3: mov eax, dword [ebp+16]
     add eax, 112
     jmp .L8
.L4: mov eax, dword [ebp+16]
     add eax, dword [ebp+12]
     sal eax, 2
.L8:
  . . .
```

```
int f(int a, int b, int c) {
   int res = 0;

   switch (a) {
   case _:
      c = _______;
   case _:
      res = _______;
      break;
   case _:
   case _:
      res = _______;
      break;
   case _:
      res = _______;
      break;
   default:
      res = _______;
   }

   return res;
}
```

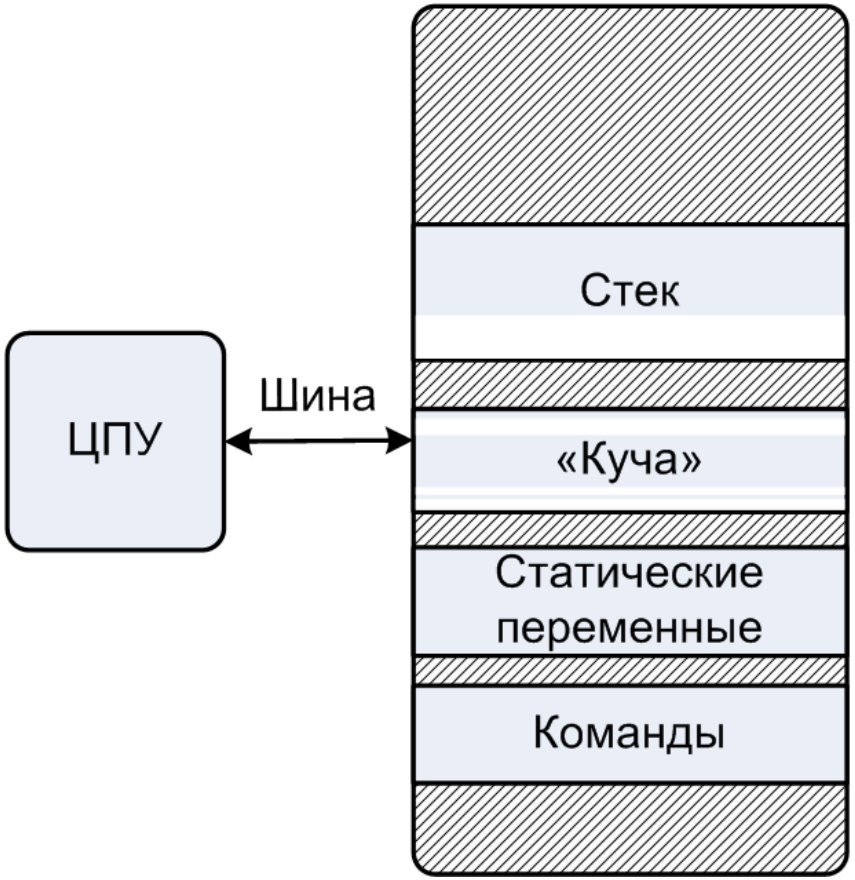
Промежуточные итоги: передача управления

- 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

Распределение памяти

- Модели памяти в языке Си
  - Автоматическая
  - Статическая
  - Динамическая

- Секции кода и данных в языке ассемблера
  - `.text`
  - `.data`, `.bss`
  - `.rodata`



```
-bash-2.05b$ ./build_asm.sh hello.asm
-bash-2.05b$ objdump -h hello
```

```
%include "io.inc"
section .rodata
  msg db `Hello, world!\n`, 0
section .text
  global CMAIN
CMAIN:
  PRINT_STRING msg
  xor eax, eax
  ret
```

```
hello:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
...
 11 .text         000001cc  08048310  08048310  00000310  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
...
 13 .rodata       00000017  080484f8  080484f8  000004f8  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
...
 15 .data         00000008  08049514  08049514  00000514  2**2
                  CONTENTS, ALLOC, LOAD, DATA
...
 21 .bss          0000000c  08049610  08049610  00000610  2**2
                  ALLOC
...
```

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

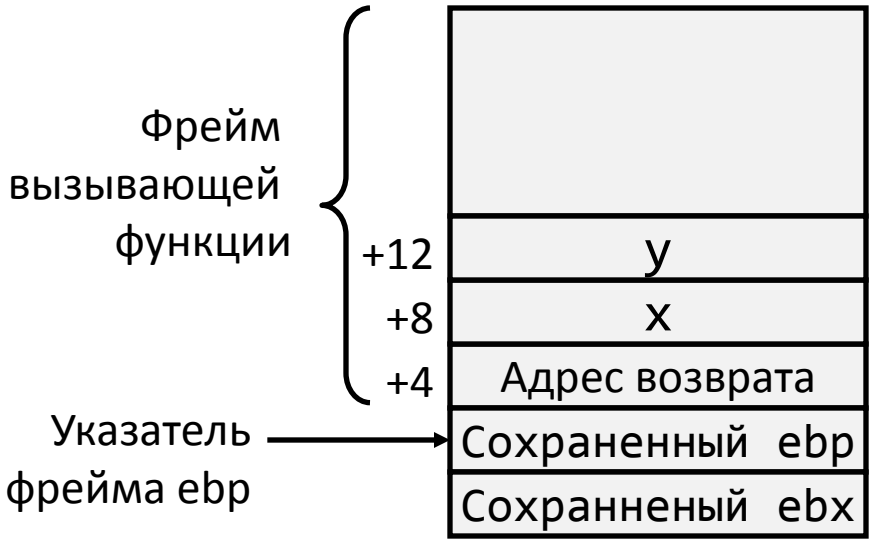
Intel	Асм.	байты	Си
byte	b	1	[unsigned] char
word	w	2	[unsigned] short
double word	d	4	[unsigned] int
quad word	q	8	[unsigned] long long int

- 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

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;
}
```

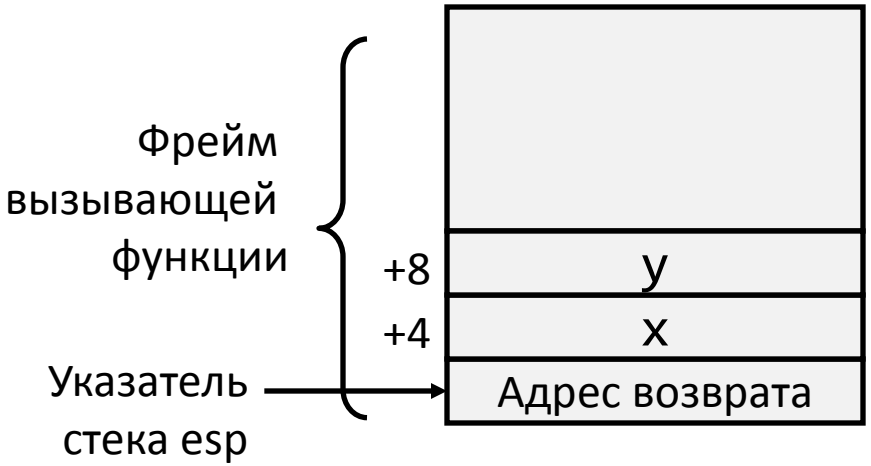
```
-bash-2.05b$ gcc -O2 -o xorcopy xorcopy.c
-bash-2.05b$ ./xorcopy
Before x = 0, y = 1
After  x = 1, y = 0
-bash-2.05b$ gcc -O0 -o xorcopy xorcopy.c
-bash-2.05b$ ./xorcopy
Before x = 0, y = 1
After  x = 0, y = 0
```

Пример3: правильный обмен с использованием команд XOR

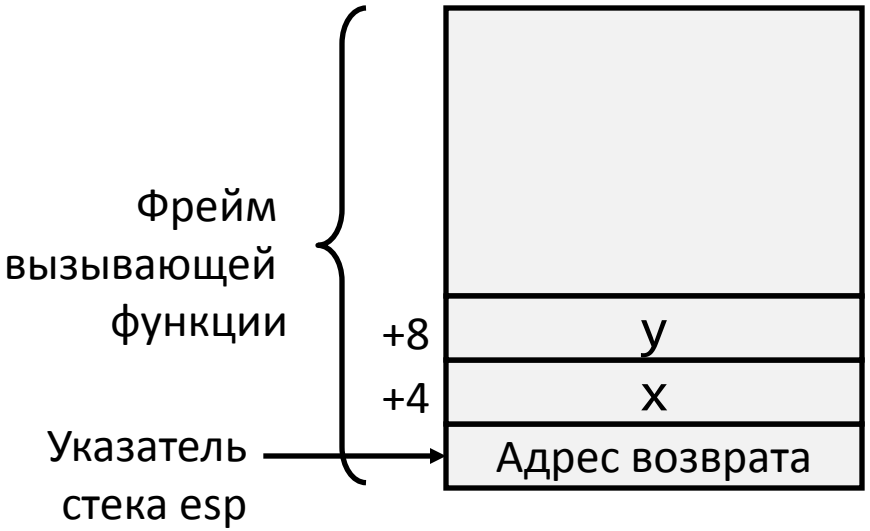
```
void exchange(int *x, int *y) {
   if (x == y) {
      return;
   }

   *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) {
        ???
}
```

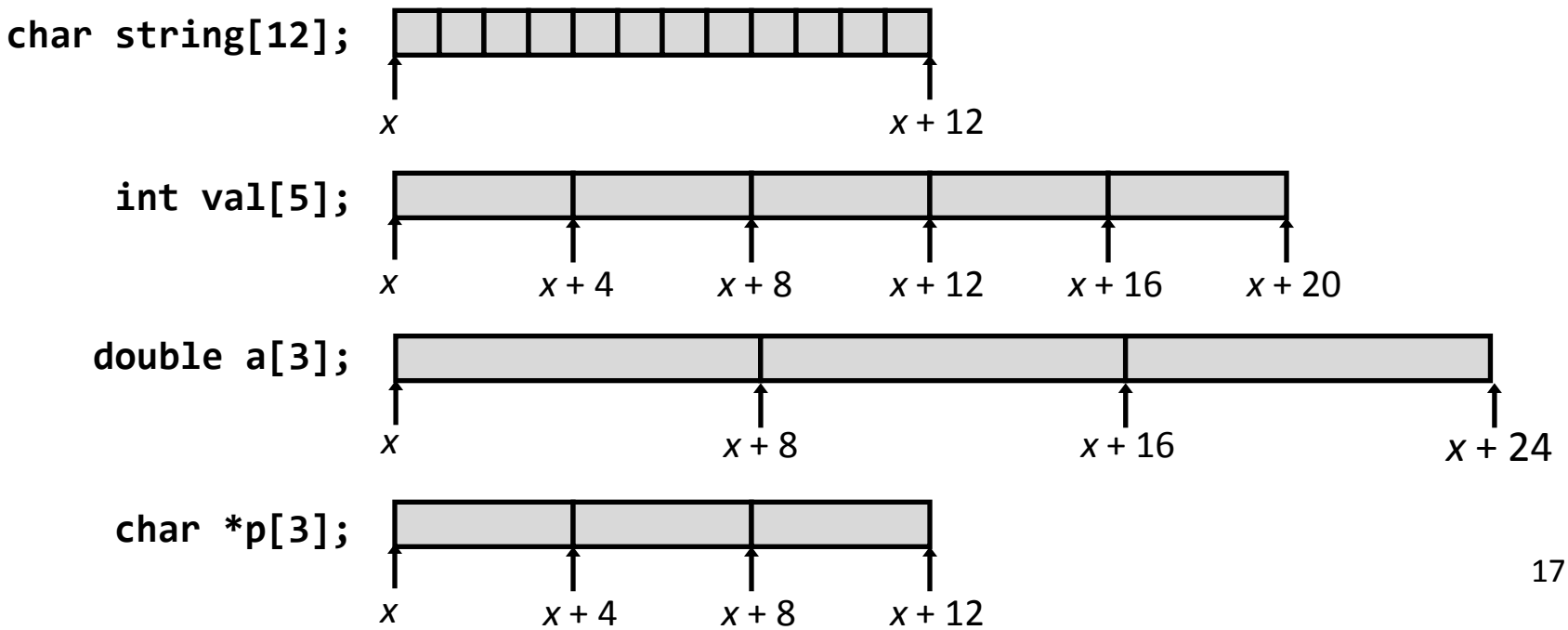
```
exchange:
   ... ; пролог функции
   mov edi, [ebp + 8]   ; (1)
   mov edx, [ebp + 12]  ; (2)
   mov ecx, [ebp + 16]  ; (3)
   mov ebx, [edx]       ; (4)
   mov esi, [ecx]       ; (5)
   mov eax, [edi]       ; (6)
   mov [edx], eax       ; (7)
   mov [ecx], ebx       ; (8)
   mov [edi], esi     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

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

Массивы – размещение в памяти

`T A[L];`

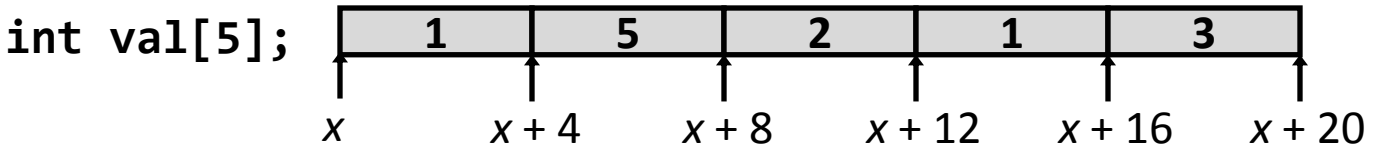
- Массив элементов типа **T**, размер массива – **L**
- Массив располагается в непрерывном блоке памяти размером **`L * sizeof(T)`** байт



Доступ к элементам массива

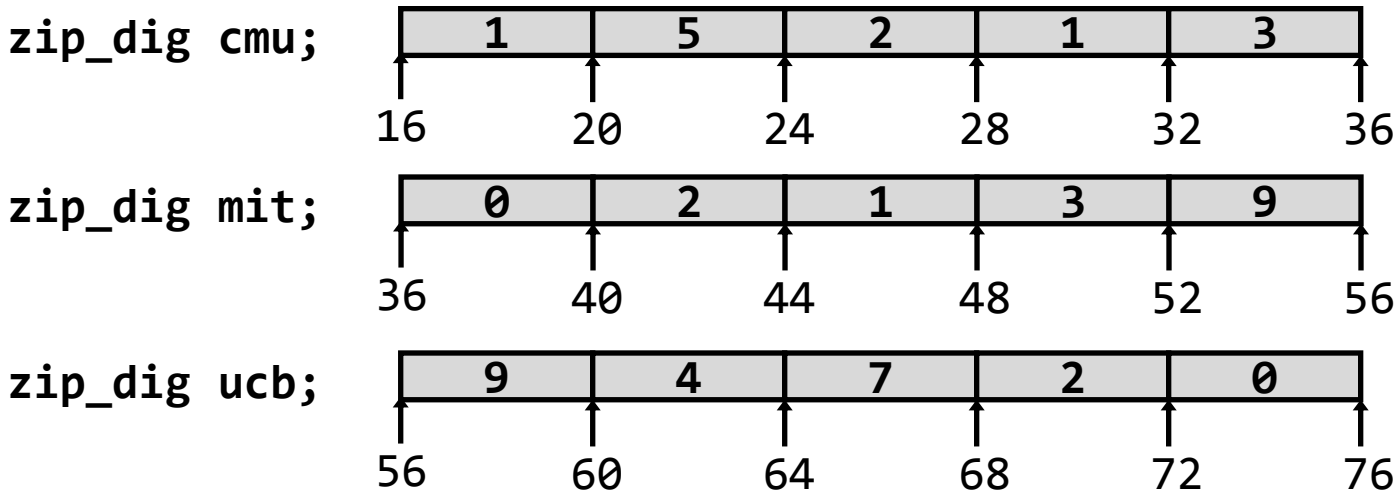
`T A[L];`

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



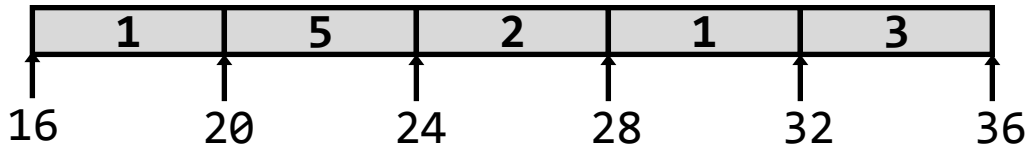
- Задачи …

```
#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
```