© 2016 МГУ/ВМК/СП

Лекция 0xB

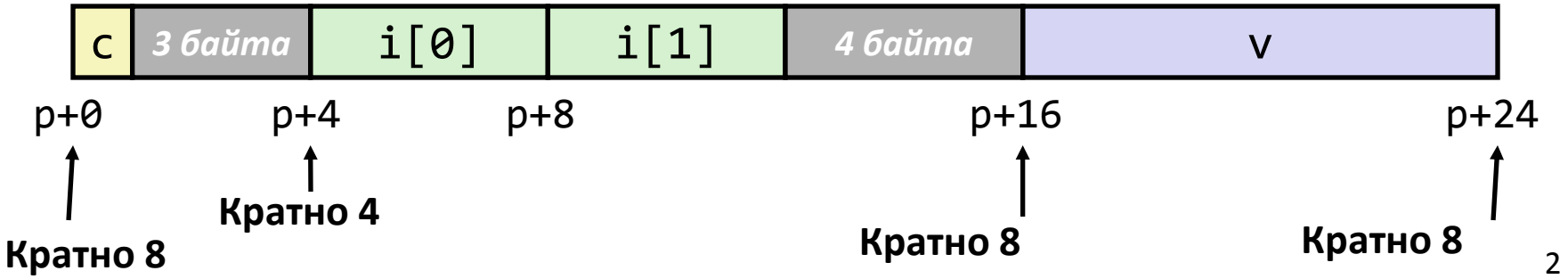
16 марта

© 2016 МГУ/ВМК/СП

Выполнение правил выравнивания для полей

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

- Внутри структуры
  - Выравнивание должно выполняться для каждого поля
- Размещение всей структуры
  - Для каждой структуры определятся требование по выравниванию в **K** байт
    - **K** = Наибольшее выравнивание среди всех полей
  - Начальный адрес структуры и ее длина должны быть кратны **K**
- Пример (для Windows или x86-64):
  - K = 8, из-за присутствия поля типа double

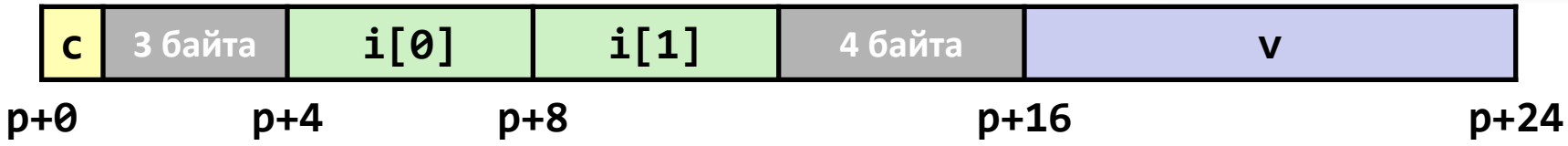


© 2016 МГУ/ВМК/СП

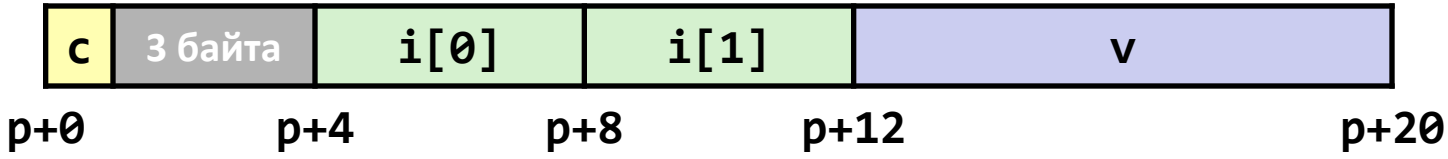
Различные соглашения о выравнивании

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

- x86-64 или IA-32 Windows:
  - K = 8, из-за наличия поля типа **`double`**



- IA-32 Linux
  - K = 4; `double` рассматривается аналогично 4-байтным типам данных

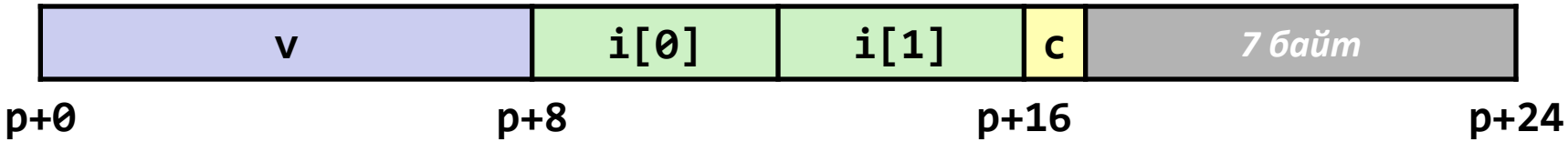


© 2016 МГУ/ВМК/СП

Выравнивание всей структуры

```
struct S2 {
  double v;
  int i[2];
  char c;
} *p;
```

- Определяется требование к выравниванию в K байт
- Общий размер структуры должен быть кратен K

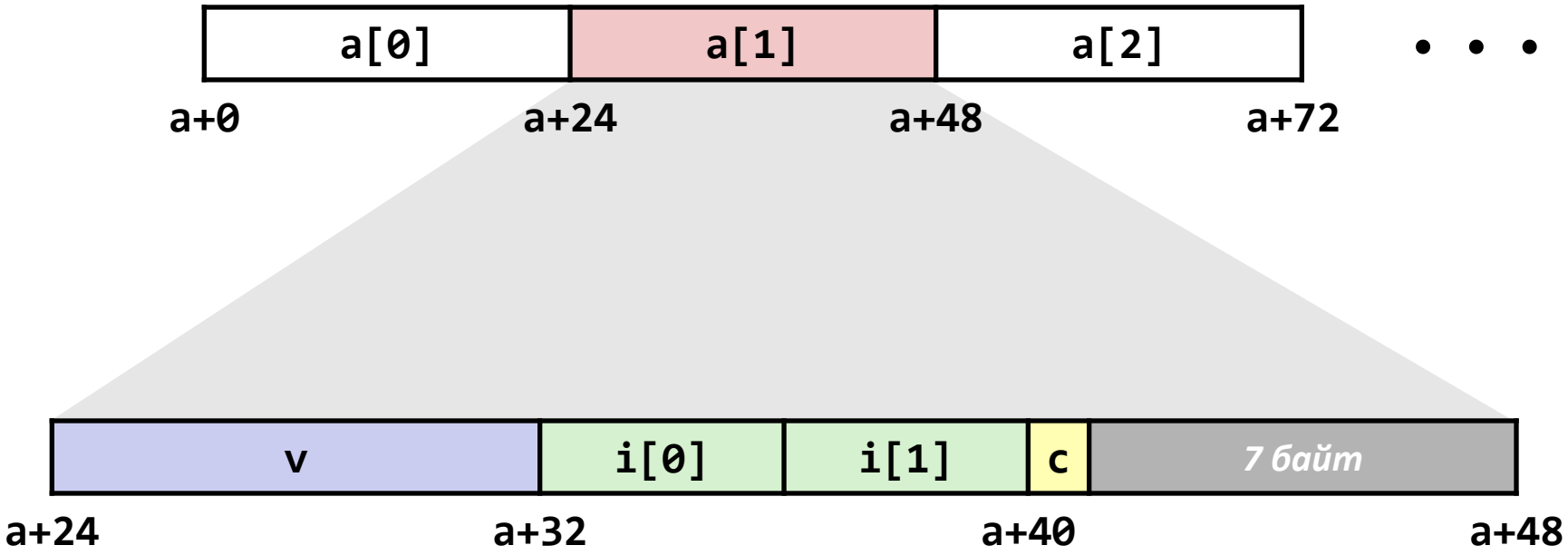


© 2016 МГУ/ВМК/СП

Массивы структур

- Размер всей структуры кратен K
- Для каждого элемента массива производится выравнивание

```
struct S2 {
  double v;
  int i[2];
  char c;
} a[10];
```

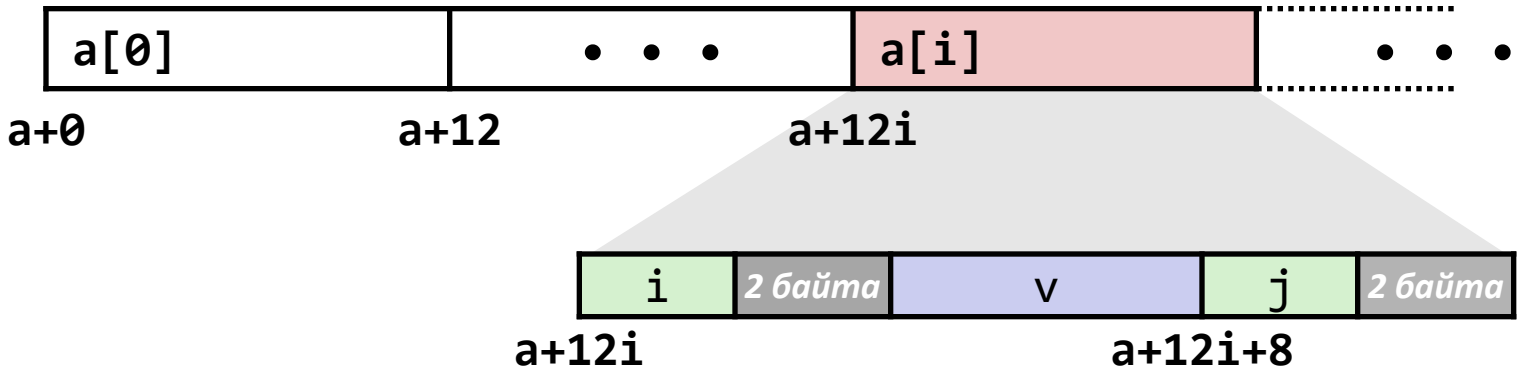


© 2016 МГУ/ВМК/СП

Доступ к элементам массива

- Вычисляем смещение в массиве
  - Вычисляем sizeof(S3), учитывая пропуски
- Вычисляем смещение внутри структуры
  - Поле j расположено со смещением 8 внутри структуры

```
struct S3 {
  short i;
  float v;
  short j;
} a[10];
```




```
short get_j(int idx) {
  return a[idx].j;
}
```

```
; eax = idx
  lea eax, [eax + 2 * eax] ; 3*idx
  movsx eax, word [a + 4 * eax + 8]
```

© 2016 МГУ/ВМК/СП

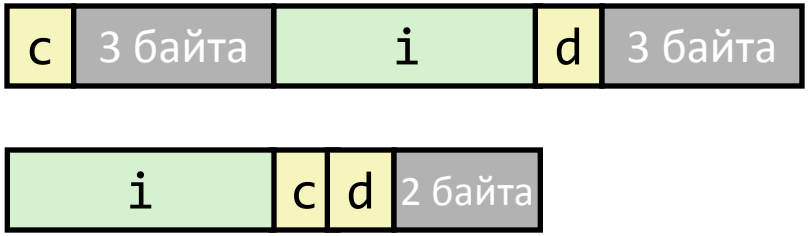
Как сохранить место

- Размещаем большие типы первыми

```
struct S4 {
  char c;
  int i;
  char d;
} *p;
```

```
struct S5 {
  int i;
  char c;
  char d;
} *p;
```

- Результат (K=4)

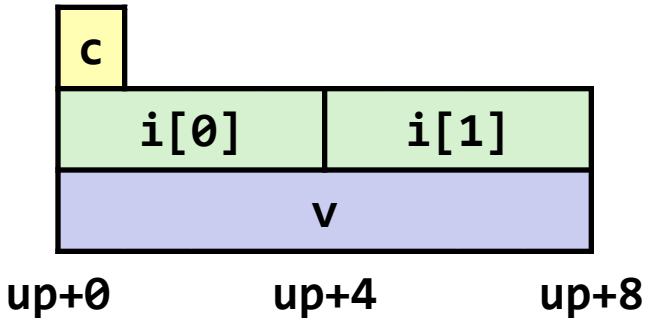


© 2016 МГУ/ВМК/СП

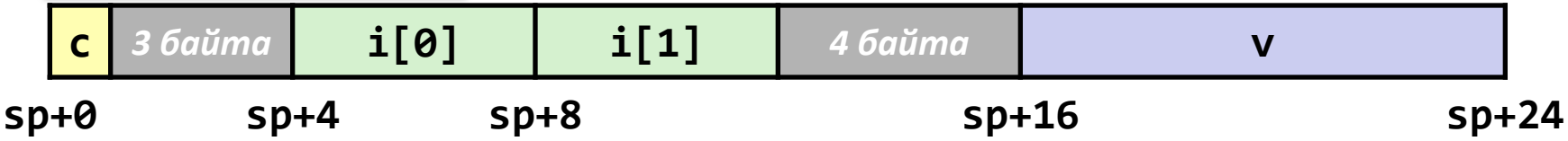
Размещение объединений

- Память выделяется исходя из размеров максимального элемента
- Используется только одно поле

```
union U1 {
  char c;
  int i[2];
  double v;
} *up;
```




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



© 2016 МГУ/ВМК/СП

Пример: двоичное дерево

```
struct NODE_S {
   struct NODE_S *left;
   struct NODE_S *right;
   double data;
};
```

В чем ошибка ?

```
union NODE_U {
   struct {
      union NODE_U *left;
      union NODE_S *right;
   } internal;
   double data;
};
```

© 2016 МГУ/ВМК/СП

Пример: двоичное дерево

```
typedef enum {N_LEAF, N_INTERNAL} nodetype_t;


struct NODE_T {
   nodetype_t type;
   union NODE_U {
      struct {
         struct NODE_T *left;
         struct NODE_T *right;
      } internal;
      double data;
   } info;
};
```

sizeof(**struct NODE_T**) ?

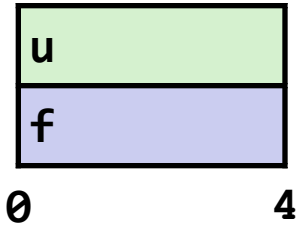
Какие смещения у полей?

IA-32 Linux / IA-32 Windows

© 2016 МГУ/ВМК/СП

Использование объединений для доступа к отдельным битам

```
typedef union {
  float f;
  unsigned u;
} bit_float_t;
```




```
float bit2float(unsigned u) {
  bit_float_t arg;
  arg.u = u;
  return arg.f;
}
```

```
unsigned float2bit(float f) {
  bit_float_t arg;
  arg.f = f;
  return arg.u;
}
```

Тоже самое, что и `(float) u` **?**

Тоже самое, что и `(unsigned) f` **?**

© 2016 МГУ/ВМК/СП

Порядок байт

- Основная идея
  - short/long/double хранятся в памяти как последовательности из 2/4/8 байт
  - Где именно расположен старший (младший) байт?
  - Может являться проблемой при пересылке двоичных данных между машинами разной архитектуры
- Big-endian / от старшего к младшему
  - Старший байт имеет наименьший адрес
  - Sparc (до V8)
- Little-endian / от младшего к старшему
  - Младший байт имеет наименьший адрес
  - Intel x86 (IA-32)
- Переключаемый порядок байт
  - ARM, PowerPC, Alpha, SPARC V9, MIPS, PA-RISC и IA-64

© 2016 МГУ/ВМК/СП

Пример

```
union {
  unsigned char c[8];
  unsigned short s[4];
  unsigned int i[2];
  unsigned long l[1];
} dw;
```

32 бита

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
s[0]		s[1]		s[2]		s[3]	
i[0]				i[1]			
l[0]							

64 бита

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
s[0]		s[1]		s[2]		s[3]	
i[0]				i[1]			
l[0]							

© 2016 МГУ/ВМК/СП

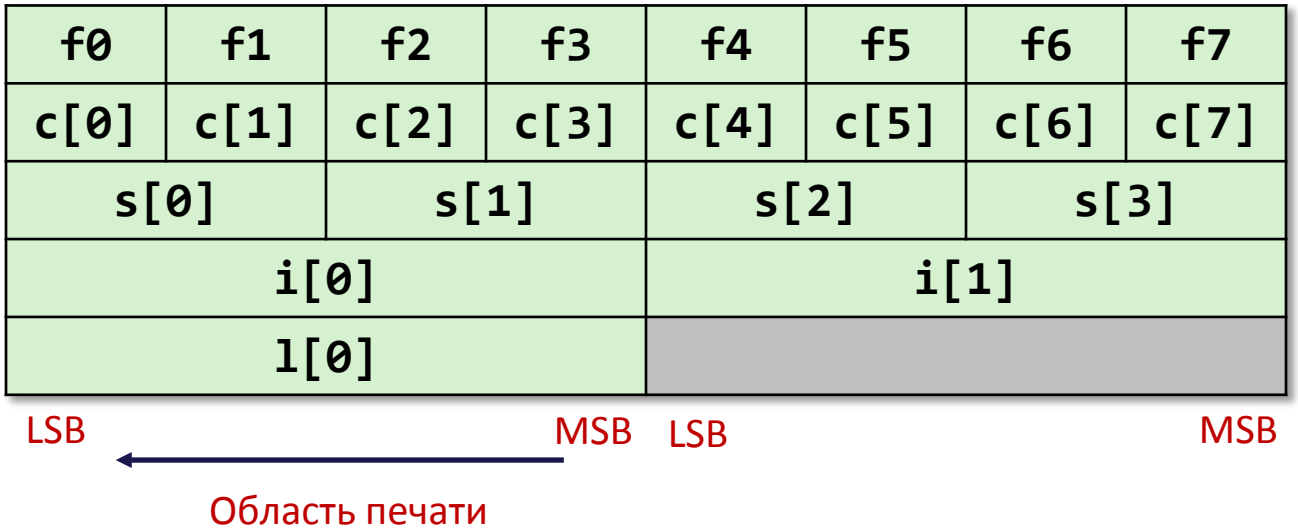
Пример (продолжение)

```
int j;
for (j = 0; j < 8; j++)
    dw.c[j] = 0xf0 + j;

printf("Characters 0-7 ==
[0x%x,0x%x,0x%x,0x%x,0x%x,0x%x,0x%x,0x%x]\n",
    dw.c[0], dw.c[1], dw.c[2], dw.c[3],
    dw.c[4], dw.c[5], dw.c[6], dw.c[7]);

printf("Shorts 0-3 == [0x%x,0x%x,0x%x,0x%x]\n",
    dw.s[0], dw.s[1], dw.s[2], dw.s[3]);

printf("Ints 0-1 == [0x%x,0x%x]\n",
    dw.i[0], dw.i[1]);

printf("Long 0 == [0x%lx]\n",
    dw.l[0]);
```

© 2016 МГУ/ВМК/СП

Порядок байт IA-32 и многих других архитектур

Порядок байт от младшего к старшему



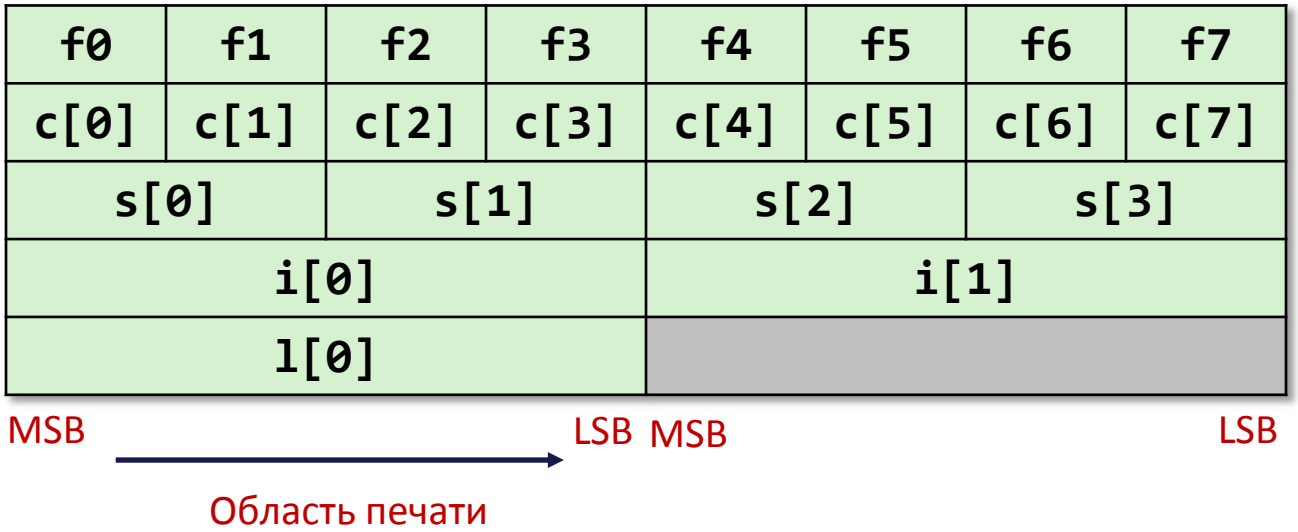
Вывод на консоль:

```
Characters 0-7 == [0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7]
Shorts     0-3 == [0xf1f0,0xf3f2,0xf5f4,0xf7f6]
Ints       0-1 == [0xf3f2f1f0,0xf7f6f5f4]
Long       0   == [0xf3f2f1f0]
```

© 2016 МГУ/ВМК/СП

Порядок байт SPARC, PPC, m68k и некоторых других архитектур

Порядок байт от старшего к младшему



Вывод на консоль:

```
Characters 0-7 == [0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7]
Shorts     0-3 == [0xf0f1,0xf2f3,0xf4f5,0xf6f7]
Ints       0-1 == [0xf0f1f2f3,0xf4f5f6f7]
Long       0   == [0xf0f1f2f3]
```

© 2016 МГУ/ВМК/СП

Порядок байт в x86-64

Порядок байт от младшего к старшему

Вывод на консоль:

```
Characters 0-7 == [0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7]
Shorts     0-3 == [0xf1f0,0xf3f2,0xf5f4,0xf7f6]
Ints       0-1 == [0xf3f2f1f0,0xf7f6f5f4]
Long       0   == [0xf7f6f5f4f3f2f1f0]
```

© 2016 МГУ/ВМК/СП

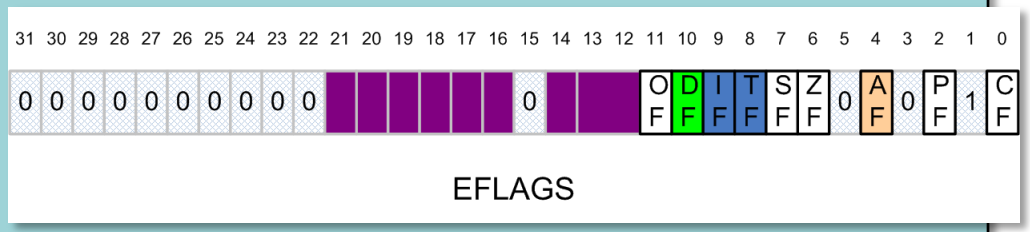
Битовые поля

- Размещение битовых полей зависит от реализации
- Битовые поля могут размещаться как справа налево, так и наоборот, в зависимости от реализации
- Битовые поля могут пересекать границы машинных слов
- Выравнивание битовых полей зависит от реализации
- К битовым полям неприменима операция & и оператор `sizeof`
- Код непереносим между различными системами

© 2016 МГУ/ВМК/СП

```c
typedef union {
   unsigned int raw;
   struct {
      int CF   : 1;
      int gap1 : 1;
      int PF   : 1;
      int gap2 : 1;
      int AF   : 1;
      int gap3 : 1;
      int ZF   : 1;
...
      int gap5 : 10;
   } fields;
} t_eflags;

int checkEflagsState(t_eflags *sw) {
   t_eflags andMask = {0};
   andMask.fields.gap2 = 1;
   andMask.fields.gap3 = 1;
   andMask.fields.gap4 = 1;
   andMask.fields.gap5 = -1;
   return !(sw->raw & andMask.raw) && (sw->raw & 2);
}
```

```asm
checkEflagsState:
   push ebp
   mov  ebp, esp
   mov  eax, dword [ebp+8]
   mov  edx, dword [eax]
   xor  eax, eax
   test edx, -4161496
     ; 11111111_11000000_10000000_00101000b
   jne  .L3
   mov  eax, edx
   shr  eax
   and  eax, 1
.L3:
   pop     ebp
   ret
```



© 2016 МГУ/ВМК/СП

Типы данных языка Си
Итоги

- Размещение переменных
  - Классы памяти: автоматическая, статическая, динамическая
  - Регистр вместо памяти (при определенных условиях)
- Массивы в языке Си
  - Непрерывная последовательность байт в памяти
  - Выравнивание всего массива удовлетворяет требования к выравниванию для каждого его элемента
  - Имя массива – указатель на его первый элемент
  - Нет никаких проверок выхода за границы
- Структуры
  - Память под поля выделяется в порядке объявления этих полей
  - Помещаются пропуски между полями и в конце всей структуры с целью выравнивания данных
- Объединения
  - Объявленные поля перекрываются в памяти
  - Способ жестокого обмана системы типов языка Си

© 2016 МГУ/ВМК/СП

Далее…

- ***Функции***
  - ***Соглашение CDECL***
    - ***Рекурсия***
  - ***Различные соглашения о вызове функций***
    - ***cdecl/stdcall/fastcall, отказ от указателя фрейма***
    - ***Соглашение вызова для x86-64***
  - ***Что происходит в Си-программе до и после функции main***
  - ***Выравнивание стека***
  - ***Переменное число параметров***
  - ***Вызов по указателю***
  - ***Переполнение буфера, эксплуатация ошибок, механизмы защиты***
- Организация динамической памяти
- Числа с плавающей точкой