© 2012 МГУ/ВМиК/СП

Лекция 11

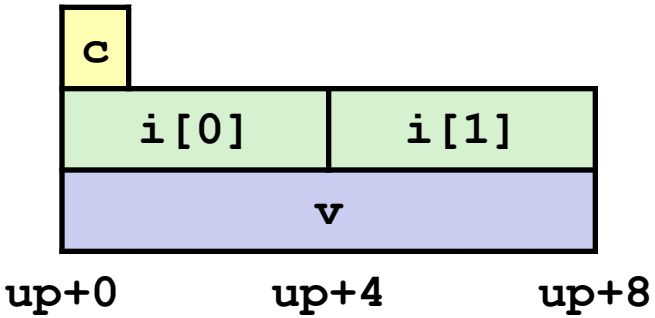
17 марта

© 2012 МГУ/ВМиК/СП

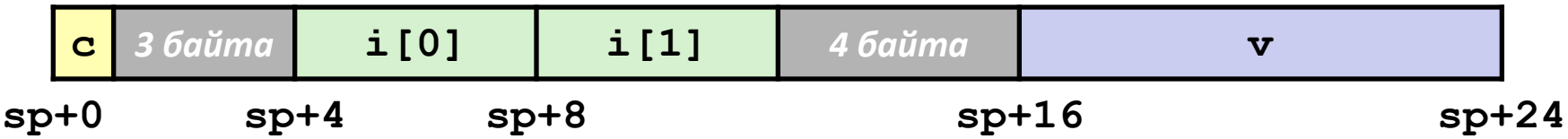
Размещение объединений

- 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;
```



© 2012 МГУ/ВМиК/СП

Пример: двоичное дерево

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

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

?

© 2012 МГУ/ВМиК/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**) ?

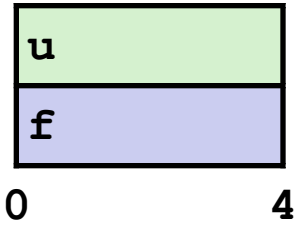
Какие смещения у полей?

IA-32 Linux / IA-32 Windows

© 2012 МГУ/ВМиК/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 **?**

© 2012 МГУ/ВМиК/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
- Обратный порядок байт
  - Старший байт имеет наименьший адрес
  - Sparc (до V8)
- Прямой порядок байт
  - Младший байт имеет наименьший адрес
  - Intel x86 (IA-32)
- Переключаемый порядок байт
  - ARM, PowerPC, Alpha, SPARC V9, MIPS, PA-RISC и IA-64

© 2012 МГУ/ВМиК/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]							

© 2012 МГУ/ВМиК/СП

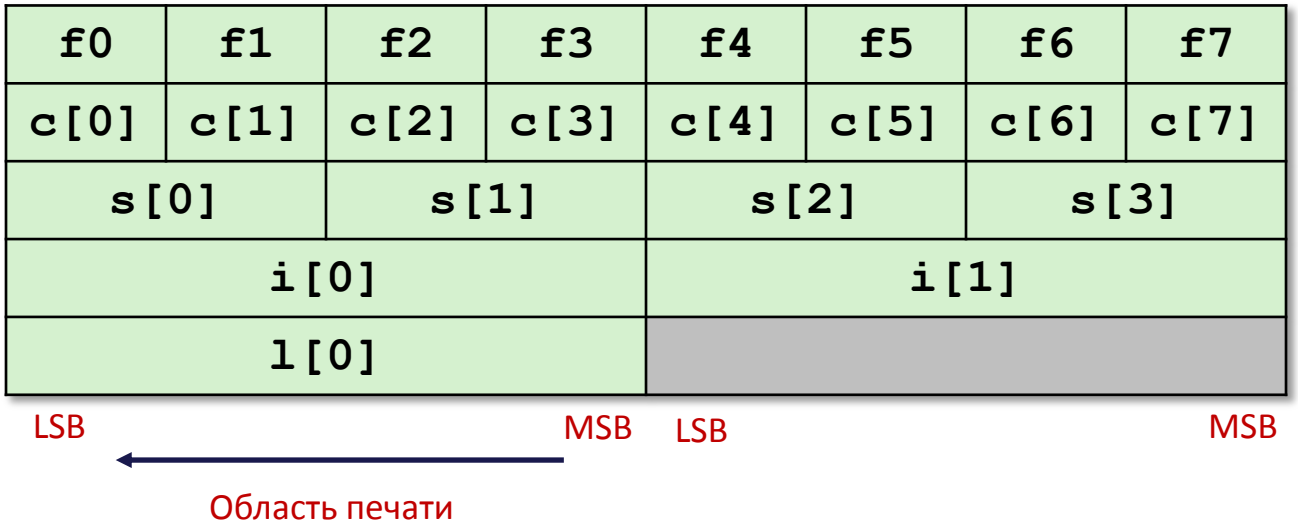
Пример (продолжение)

```
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]);
```

© 2012 МГУ/ВМиК/СП

Порядок байт IA32

Прямой порядок байт



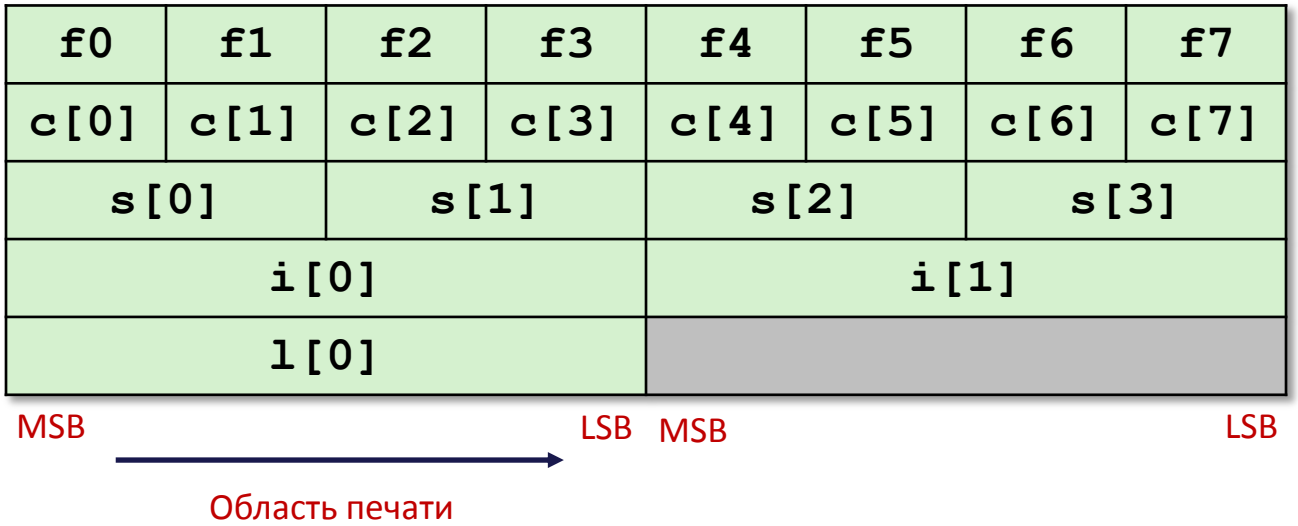
Вывод на консоль:

```
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]
```

© 2012 МГУ/ВМиК/СП

Порядок байт Sun

Обратный порядок байт



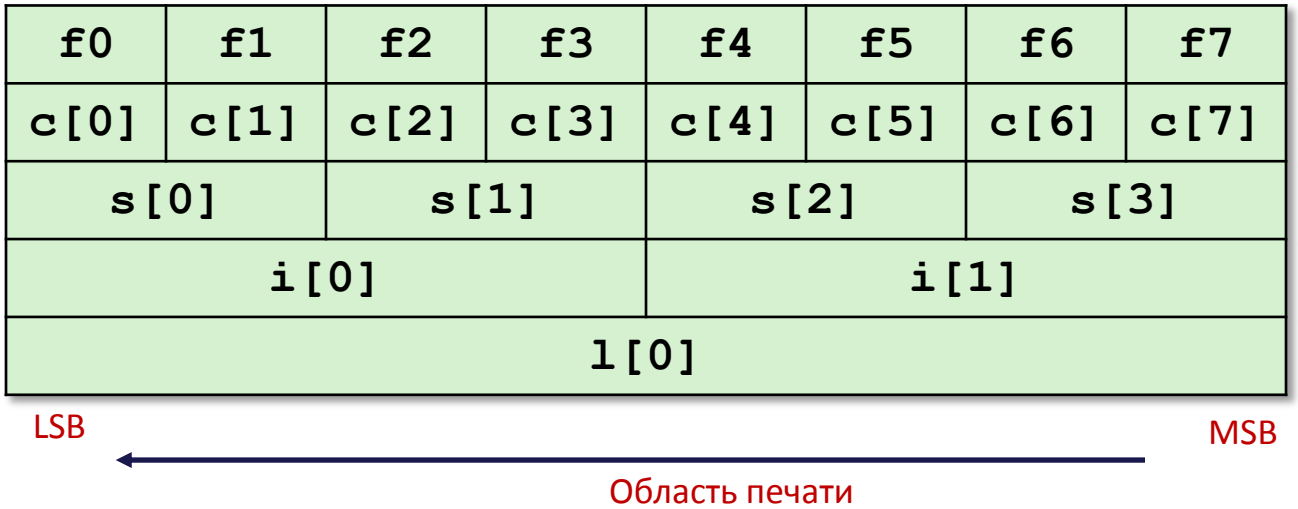
Вывод на консоль:

```
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]
```

© 2012 МГУ/ВМиК/СП

Порядок байт x86-64

Прямой порядок байт



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]
```

© 2012 МГУ/ВМиК/СП

Промежуточный итог

- 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 для с целью выравнивания
- Объединения
  - Объявленные поля перекрываются в памяти
  - Способ жестокого обмана системы типов языка Си

© 2012 МГУ/ВМиК/СП

Операции над отдельными битами

- Логические инструкции
  - AND
  - OR
  - XOR
  - NOT
- Сдвиги и вращения
  - SAR
  - SHR
  - SAL, SHL
  - ROR
  - ROL
  - RCR
  - RCL
- Битовые и байтовые
  - BT
  - BTS
  - BTR
  - BTC
  - BSF
  - BSR
  - SET*cc*
  - TEST

© 2012 МГУ/ВМиК/СП

Пример

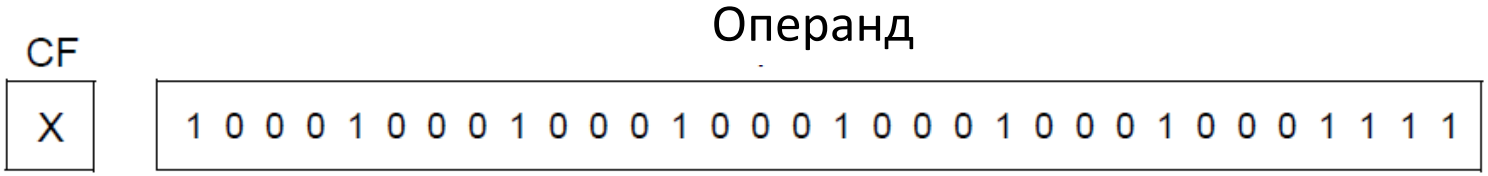
```
int pierce_arrow(int a, int b) {
   int t = ~(a | b);
   return t;
}
```

```
section .text
global pierce_arrow
pierce_arrow:
   push    ebp
   mov     ebp, esp
   mov     eax, dword [ebp+12]  ; (1)
   or      eax, dword [ebp+8]   ; (2)
   not     eax                  ; (3)
   pop     ebp
   ret
```

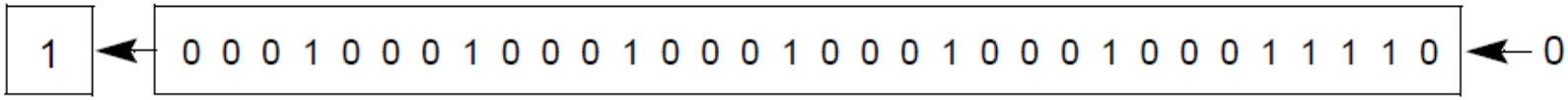
© 2012 МГУ/ВМиК/СП

Сдвиг влево

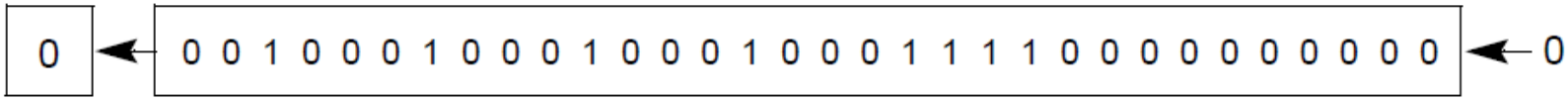
Начальное состояние



После сдвига на один разряд: SHL/SAL



После сдвига на 10 разрядов: SHL/SAL



© 2012 МГУ/ВМиК/СП

Логический сдвиг вправо

Начальное состояние

Операнд

CF

1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 1 1 1 | X

После сдвига на один разряд: SHR

0 → 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 1 1 → 1

После сдвига на 10 разрядов: SHR

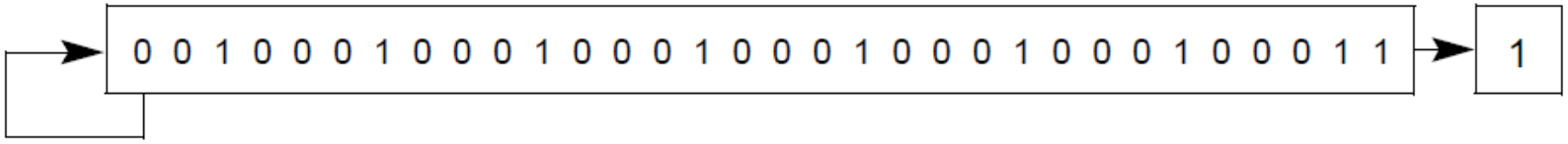
0 → 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 → 0

© 2012 МГУ/ВМиК/СП

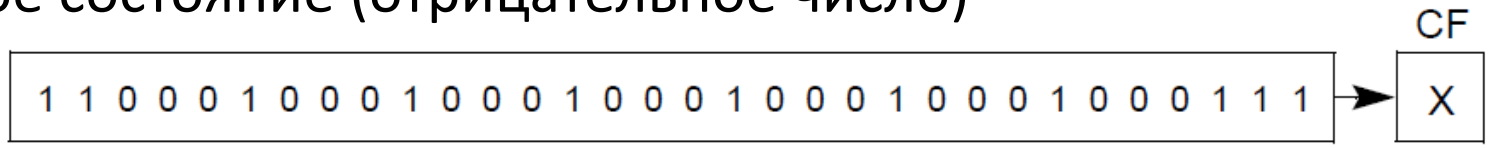
Арифметический сдвиг вправо

Начальное состояние (положительное число)

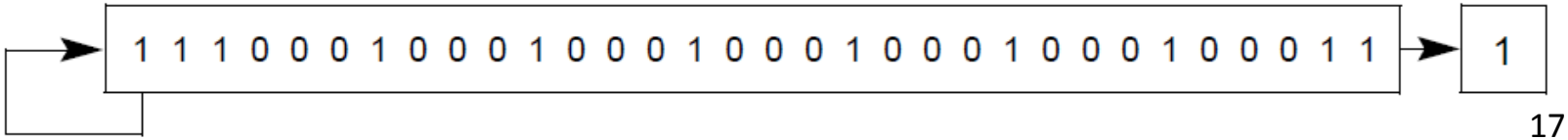
После сдвига на один разряд: SAR



Начальное состояние (отрицательное число)



После сдвига на один разряд: SAR



© 2012 МГУ/ВМиК/СП

Пример

```
char upndown(char x) {
   return (x << 8) >> 8;
}
```

Что вернет `upndown(42);` ?

© 2012 МГУ/ВМиК/СП

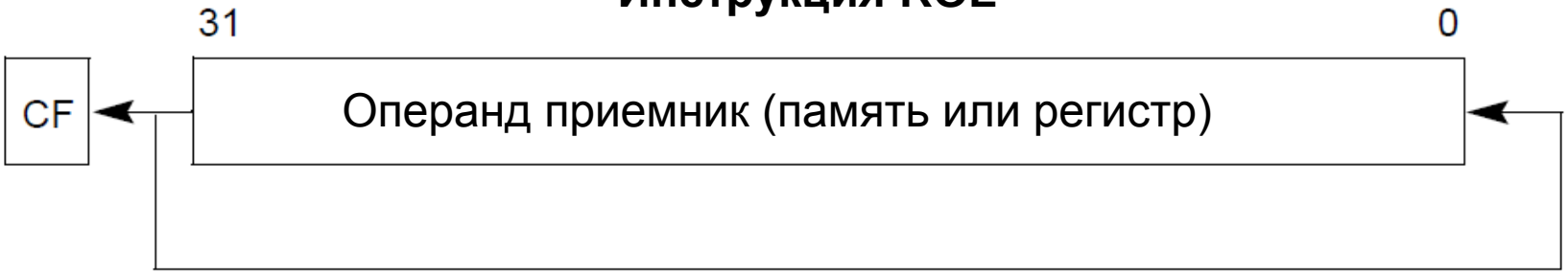
Пример

```
char upndown(char x) {
   return (x << 8) >> 8;
}
```

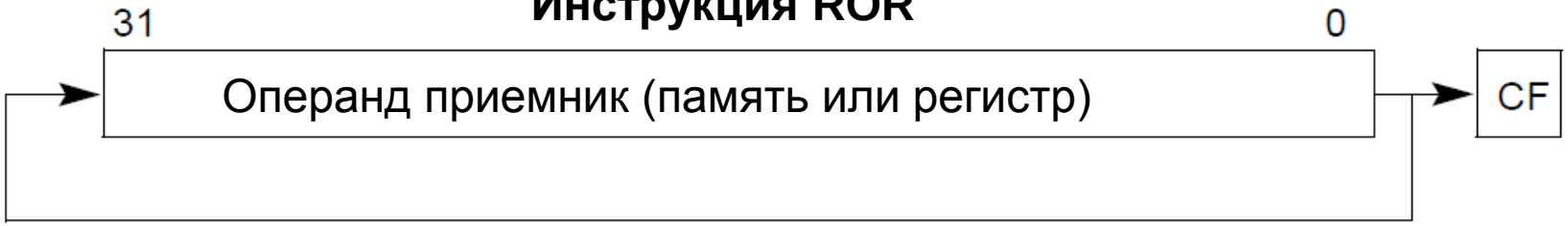
```
section .text
global upndown
upndown:
   push    ebp
   mov     ebp, esp
   movsx   eax, byte [ebp+8]
   sal     eax, 8
   sar     eax, 8
   pop     ebp
   ret
```

© 2012 МГУ/ВМиК/СП

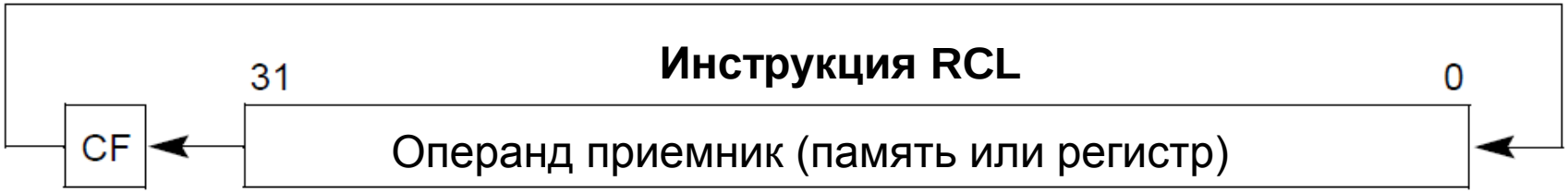
Инструкция ROL



Инструкция ROR



Инструкция RCL



Инструкция RCR

© 2012 МГУ/ВМиК/СП

Пример

```
unsigned sha256_f1(unsigned x) {
   unsigned t;
   t  = ((x >> 2)  | (x << ((sizeof(x) << 3) - 2)));  // (1)
   t ^= ((x >> 13) | (x << ((sizeof(x) << 3) - 13))); // (2)
   t ^= ((x >> 22) | (x << ((sizeof(x) << 3) - 22))); // (3)
   return t;
}
```

```
global sha256_f1
sha256_f1:
   push    ebp
   mov     ebp, esp
   mov     edx, dword [ebp+8] ; (1)
   pop     ebp                ; (2)
   mov     eax, edx           ; (3)
   mov     ecx, edx           ; (4)
   ror     eax, 13            ; (5)
   ror     ecx, 2             ; (6)
   xor     eax, ecx           ; (7)
   ror     edx, 22            ; (8)
   xor     eax, edx           ; (9)
   ret
```

© 2012 МГУ/ВМиК/СП

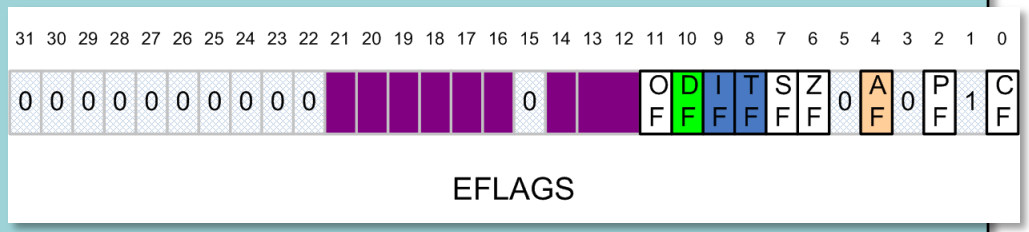
Битовые поля

- 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

© 2012 МГУ/ВМиК/СП

```
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);
}
```

```
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
```



© 2012 МГУ/ВМиК/СП

Промежуточные итоги

- Типы данных языка Си: скалярные типы, составные типы
- Машинные типы данных
- Вычисление выражений
- Арифметические, логические, битовые команды
- Управляющие операторы
- Условная и безусловная передача управления