

Лекция 6

26 февраля

Группы команд

- **Общего назначения**
- **x87 FPU**
- MMX
- SSE
- IA-32e
команды
64-разрядного режима
работы
- **Системные команды**
- Аппаратная
виртуализация
- ...

Цветом выделены группы команд, которые рассматриваются в курсе.

- Пересылка данных
- Команды двоичной арифметики
- Команды двоично-десятичной арифметики
- Логические команды
- Сдвиги и вращения
- Битовые и байтовые команды
- Передача управления
- Строковые
- Ввод/Вывод
- Явное управление EFLAGS
- Вспомогательные

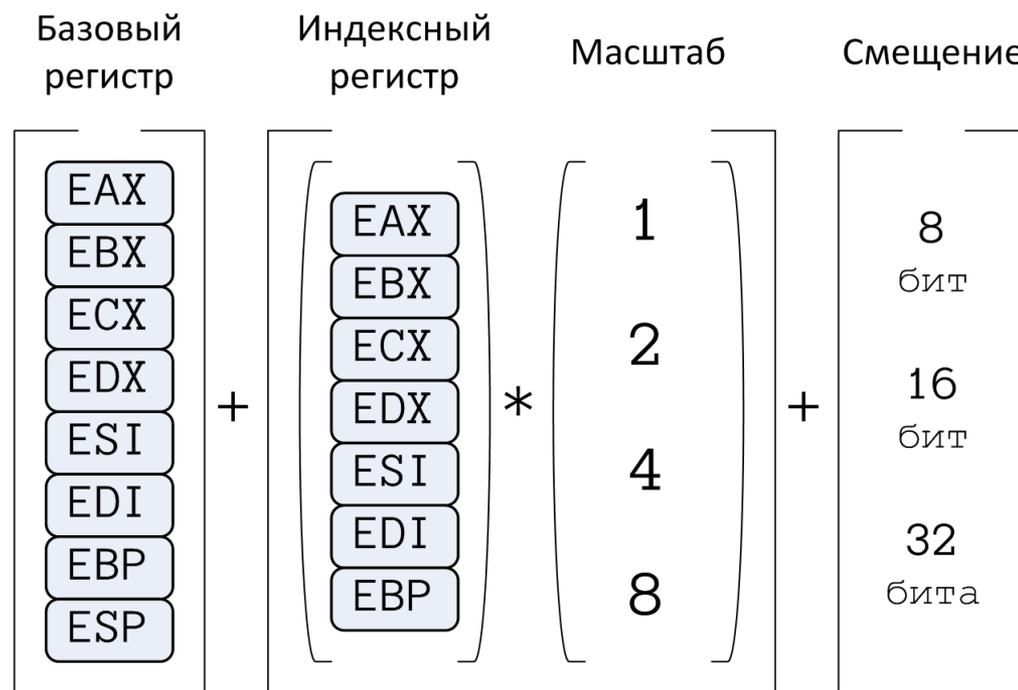
«Основные» команды IA-32

Пересылка данных	Двоичная арифметика ✓	Передача управления	Логические ✓	Сдвиги и вращения	Битовые и байтовые	Прочее
<ul style="list-style-type: none"> • MOV • XCHG • BSWAP • MOVSX • MOVZX • CDQ • CWD • CBW • PUSH • POP • CMOVCc 	<ul style="list-style-type: none"> • ADD • ADC • SUB • SBB • NEG • IMUL • MUL • IDIV • DIV • INC • DEC • CMP 	<ul style="list-style-type: none"> • JMP • Jcc • CALL • RET 	<ul style="list-style-type: none"> • AND • OR • XOR • NOT 	<ul style="list-style-type: none"> • SAR • SHR • SAL, SHL • ROR • ROL • RCR • RCL 	<ul style="list-style-type: none"> • SETcc • TEST 	<ul style="list-style-type: none"> • LEA • NOP

Красным выделены не упоминавшиеся ранее на лекциях ассемблерные инструкции. Команды двоичной арифметики и логические команды представлены в полном составе.

Общий вид адресного кода при обращении к памяти

- LEA
– r32, m



Исполнительный адрес = База + (Масштаб * Индекс) + Смещение

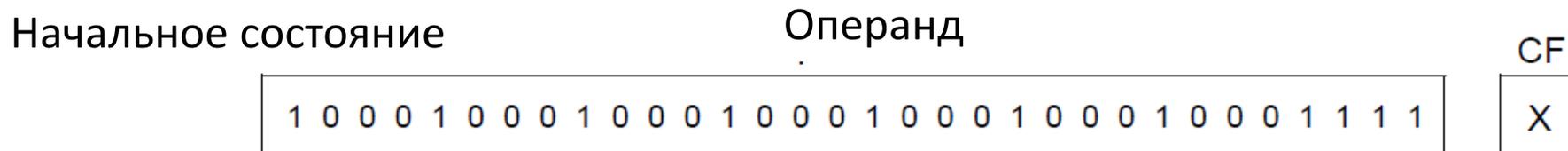
Пример

Реализация стрелки Пирса

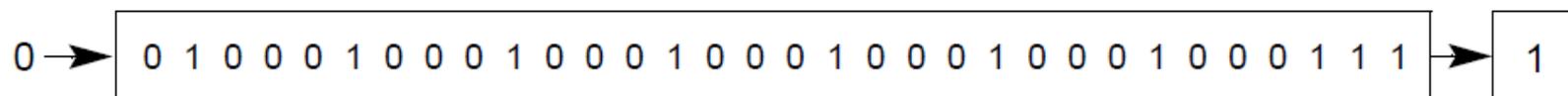
```
unsigned pierce_arrow(unsigned a, unsigned b) {  
    int t = ~(a | b);  
    return t;  
}
```

```
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
```

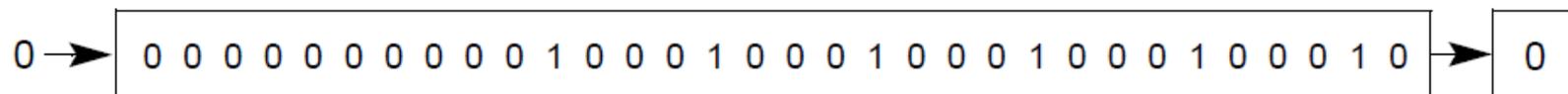

Логический сдвиг вправо



После сдвига на один разряд: SHR



После сдвига на 10 разрядов: SHR

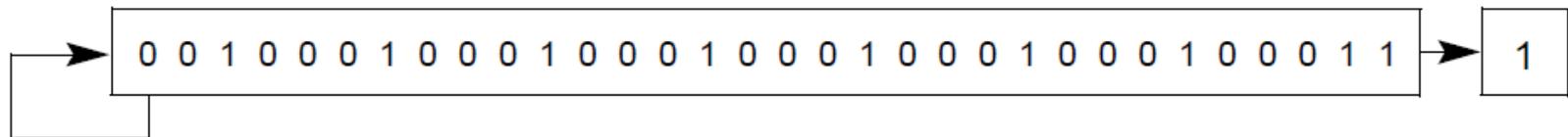


Арифметический сдвиг вправо

Начальное состояние (положительное число)



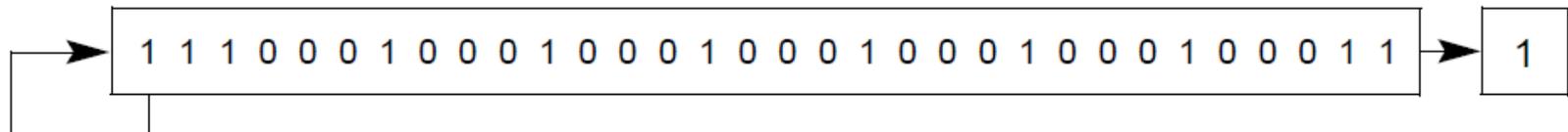
После сдвига на один разряд: SAR



Начальное состояние (отрицательное число)



После сдвига на один разряд: SAR



Пример

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

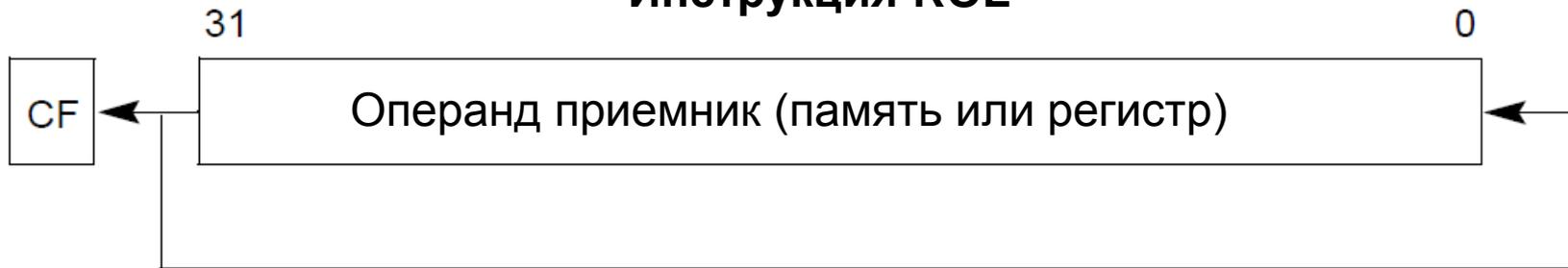
Что вернет `upndown(42);` ?

Пример

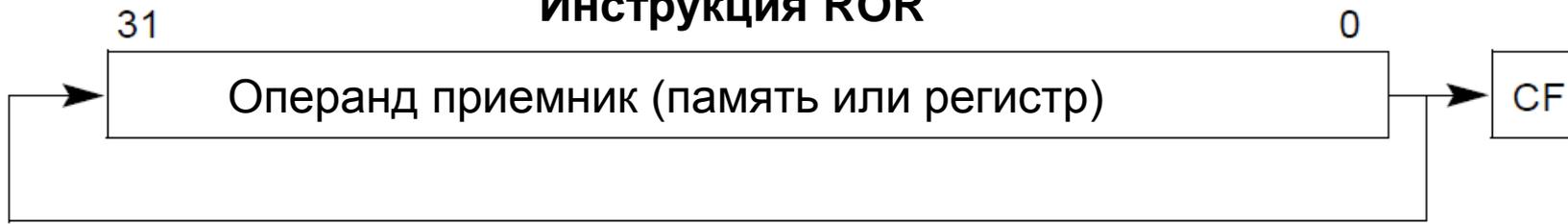
```
char upndown(char x) {  
    return (x << 8) >> 8;  
}
```

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

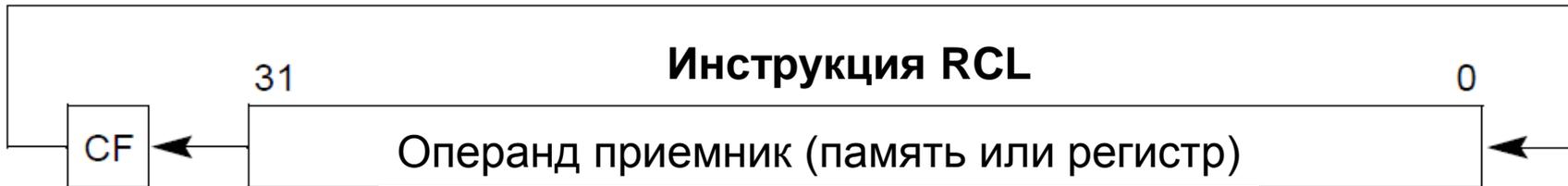
Инструкция ROL



Инструкция ROR



Инструкция RCL



Инструкция RCR



Пример: фрагмент кода криптографической хеш-функции

В 2008 году Intel были заявлены команды, поддерживающие шифрование AES: AESENC, AESDEC, AESENCLAST, AESKEYGENASSIST, ...

```
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;
}
```

Криптографические
хеш-функции $H(X)$:

- Необратимость
 $H(X) = m$, m – задано
- Стойкость к коллизиям
первого рода
 $H(M) = H(N)$, M – задано
- Стойкость к коллизиям
второго рода
 $H(M_1) = H(M_2)$

```
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
```

Обратная задача

```
static int a, b, c, d;
```

```
???
```

```
mov     eax, dword [b] ; (1)
mov     edx, dword [c] ; (2)
or      al, -1         ; (3)
sal     eax, 3         ; (4)
add     edx, eax       ; (5)
mov     dword [a], eax ; (6)
mov     eax, edx       ; (7)
sar     edx, 31        ; (8)
idiv   dword [d]      ; (9)
mov     dword [c], eax ; (10)
```

«Основные» команды IA-32

Пересылка данных	Двоичная арифметика ✓	Передача управления	Логические ✓	Сдвиги и вращения	Битовые и байтовые	Прочее
<ul style="list-style-type: none"> • MOV • XCHG • BSWAP • MOVSX • MOVZX • CDQ • CWD • CBW • PUSH • POP • CMOVCc 	<ul style="list-style-type: none"> • ADD • ADC • SUB • SBB • NEG • IMUL • MUL • IDIV • DIV • INC • DEC • CMP 	<ul style="list-style-type: none"> • JMP • Jcc • CALL • RET 	<ul style="list-style-type: none"> • AND • OR • XOR • NOT 	<ul style="list-style-type: none"> • SAR • SHR • SAL, SHL • ROR • ROL • RCR • RCL 	<ul style="list-style-type: none"> • SETcc • TEST 	<ul style="list-style-type: none"> • LEA • NOP

Красным выделены не упоминавшиеся ранее на лекциях ассемблерные инструкции. Команды двоичной арифметики и логические команды представлены в полном составе.

Сложение 64 разрядных чисел на 32 разрядных регистрах

```
long long f1(long long a, long long b) {  
    long long c;  
    c = a + b;  
    return c;  
}
```

; начало функции пропущено

```
mov     eax, dword [ebp+16] ; (1)
```

```
mov     edx, dword [ebp+20] ; (2)
```

```
add     eax, dword [ebp+8]  ; (3)
```

```
adc     edx, dword [ebp+12] ; (4)
```

; конец функции пропущен

Вычитание 64 разрядных чисел на 32 разрядных регистрах

```
long long f1(long long a, long long b) {  
    long long c;  
    c = a - b;  
    return c;  
}
```

; начало функции пропущено

```
mov     eax, dword [ebp+8] ; (1)
```

```
mov     edx, dword [ebp+12] ; (2)
```

```
sub     eax, dword [ebp+16] ; (3)
```

```
sbb     edx, dword [ebp+20] ; (4)
```

; конец функции пропущен

Умножение 64 разрядных чисел на 32 разрядных регистрах

```
long long f2(long long a,  
             long long b) {  
    long long c;  
    c = a * b;  
    return c;  
}
```

```
global f2  
f2:  
    push    ebp  
    mov     ebp, esp  
    push   ebx  
    mov     eax, dword [ebp+8]  
    mov     edx, dword [ebp+16]  
    mov     ecx, dword [ebp+20]  
    mov     ebx, dword [ebp+12]  
    imul   ecx, eax  
    imul   ebx, edx  
    mul    edx  
    add    ecx, ebx  
    add    edx, ecx  
    pop    ebx  
    pop    ebp  
    ret
```

Далее...

- Арифметика
- Операции над машинными словами
- **Реализация управляющих операторов языка Си**
 - Условная передача данных
 - Организация циклов
 - Оператор switch