Лекция 12
Язык ассемблера x86-2

Регистры процессора

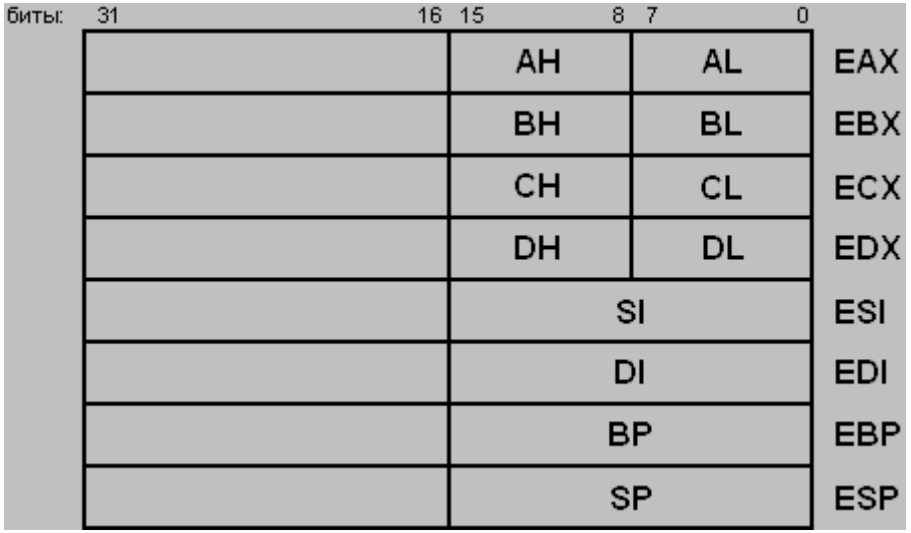
- Регистры процессора – ячейки памяти, находящиеся в процессоре
  - Очень быстрые
  - Их мало или очень мало
  - Несколько функциональных групп регистров
- Вся совокупность регистров – регистровый файл (register file)
- Регистры имеют “индивидуальные” имена

Регистры общего назначения

- General purpose register (GPR)
- Используются для:
  - Хранения аргументов для операций
  - Сохранения результатов операций
  - Хранения адреса или индекса для косвенного обращения к памяти или косвенных переходов
  - Размещения наиболее часто используемых переменных
- X86 – 8 32-битных регистров общего назначения

Регистры общего назначения

- 32-битные %eax, … %esp
- %esp – указатель стека
- 16-битные %ax… %bp
- 8-битные %al, …
- %ebp обычно указатель кадра стека, но может быть GPR
- Регистры неоднородны – в некоторых инструкциях используются фиксированные регистры



Инструкция процессора

- Инструкция – минимальное цельное действие, которое может выполнить процессор
- Выполняется или не выполняется целиком, то есть не изменяет состояние процессора “частично” по сравнению с описанием (ну почти – см. Meltdown)
- Инструкции исполняются последовательно, кроме инструкций передачи управления

Передача управления

- Безусловная передача управления:
  JMP ADDR/LABEL
- Условная передача управления
  J*cc* ADDR/LABEL
  (рассмотрим далее)
- Вызов подпрограммы
  CALL ADDR/LABEL
- Программное прерывание
  INT NUM
  (примерно как CALL)

Оперативная память

- Как правило при наличии ОС мы работаем с виртуальной памятью (будем рассматривать позднее)
- Ядро ОС или программы на bare metal работают с физической памятью
- Память в x86/x64 и многих других процессорах (ARM, MIPS, …) адресуется побайтово, т. е. каждые 8 бит (в Си/Си++ [unsigned/signed] char) имеют свой адрес
- На уровне процессора адрес – это 32/64 битное целое число

Стек

- Стек размещается в оперативной памяти (не в процессоре)
- Максимальный размер стека ограничен:
  - Структурой адресного пространства
  - Размером памяти
  - Настройками ОС
- Стек на x86/x64, ARM, MIPS, PPC и т. п. **растет вниз**, то есть в сторону уменьшения адресов

Стек на x86/x64

- Растет в сторону уменьшения адресов
- %esp (%rsp на x64) – это адрес первой используемой ячейки стека
- Работа ведется словами, т. е. 32 бита на x86 и 64 бита на x64 (меньше – нельзя)
- Например занесение значения регистра eax:
  push %eax => sub $4,%esp; mov %eax,(%esp)
- Извлечение в eax
  pop %eax => mov (%esp), %eax; add $4, %esp

Подпрограммы

- Вызов подпрограммы: CALL
  CALL LABEL
  это (схематически)
  ```
  push %eip   // сохранение адреса возврата
  jmp LABEL  // безусловный переход
  ```
- Завершение подпрограммы: RET
  RET
  это (схематически)
  ```
  pop %eip
  ```
- Все прочие регистры процессора остаются без изменений
- Передача параметров/возврат значения реализуется программно – описывается т. н. соглашениями о вызовах (calling convention)

Методы адресации

- Возможные типы аргументов операции определяются поддерживаемыми процессором методами адресации
- Методы адресации:
  - Регистровый – указывается имя регистра
    movl %esp, %ebp
  - Непосредственный (immediate) – аргумент задается в инструкции – знак $
    movb **$16**, %cl
  - Прямой (direct) – адрес ячейки памяти задается в инструкции
    movl %eax, var1

Методы адресации

- Методы адресации (продолжение)
  - Косвенный (indirect) только для jmp и call
    call *%eax
  - Относительный (relative) только для jmp и call
    jmp loop
    вычисляется смещение относительно текущего значения EIP и адреса метки, в инструкции сохраняется смещение; при выполнении перехода восстанавливается абсолютный адрес перехода

Методы адресации

- Общий вид обращения к памяти:
  OFFSET(BREG, IREG, SCALE)
  адрес вычисляется по формуле:
  BREG + OFFSET + IREG * SCALE
- BREG, IREG – GPR
- SCALE – {1, 2, 4, 8}, по умолчанию 1

Обращения к памяти

- Примеры:
  (%eax) // адрес находится в %eax
  16(%esi) // адрес равен %esi + 16
  array(,%eax) // адрес равен array + %eax
  array(,%eax,4) // адрес равен array + %eax*4
  (%ebx,%eax,2) // адрес: %ebx + %eax * 2
  -4(%ebx,%eax,8) // адрес: %ebx-4+%eax*8

Преобразования целых

- Расширение нулями:
  ```
  movzbl   var, %eax   // 8 → 32 бита
  movzwl   var, %eax   // 16 → 32 бита
  ```
- Расширение знаковым битом:
  ```
  movsbl   var, %eax
  movswl   var, %eax
  cdq                  // eax → eax:edx
  ```

Арифметика

- Арифметические инструкции:

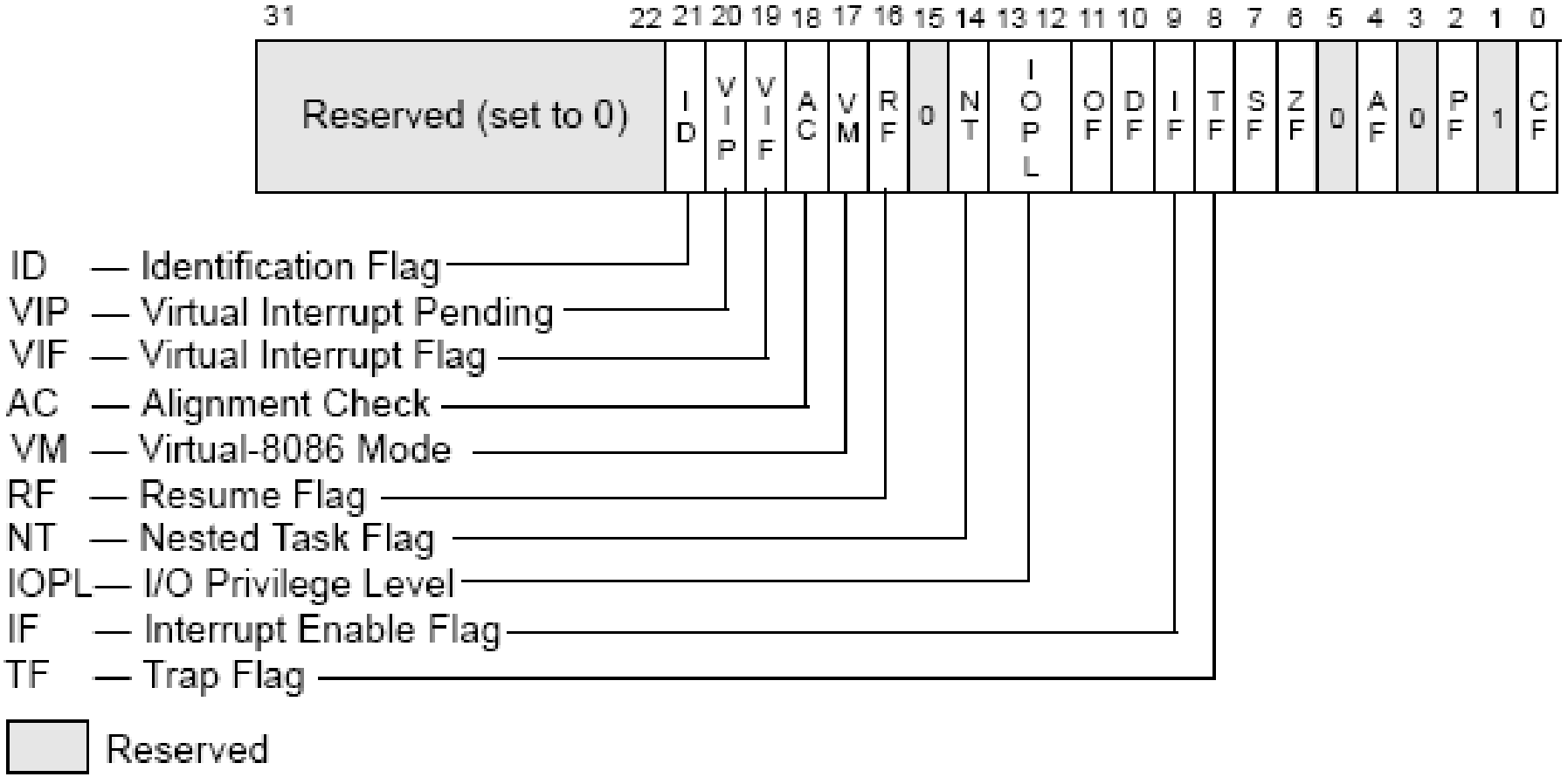
```
add  SRC, DST      // DST += SRC
sub  SRC, DST      // DST -= SRC
cmp  SRC1, SRC2    // SRC2 – SRC1
and  SRC, DST      // DST &= SRC
or SRC, DST        // DST |= SRC
xor  SRC, DST      // DST ^= SRC
test SRC1, SRC2    // SRC1 & SRC2
not  DST           // DST = ~DST
neg  DST           // DST = -DST
inc  DST           // ++DST
dec  DST           // --DST
```

Флаги результата операции

- Регистр EFLAGS содержит специальные биты-флаги результата операции
- Для x86 они называются: ZF, SF, CF, OF
  - ZF (бит 6) – флаг нулевого результата
  - SF (бит 7) – флаг отрицательного результата
  - CF (бит 0) – флаг переноса из старшего бита
  - OF (бит 11) – флаг переполнения

Регистр EFLAGS

- Нас интересуют: CF, ZF, SF, OF



Примеры

1(1) + 2(2) = 3(3), ZF=0, SF=0, CF=0, OF=0

0(0) + 0(0) = 0(0), ZF=1, SF=0, CF=0, OF=0

130(-126)+0(0)=130(-126), ZF=0,SF=1,CF=0, OF=0

130(-126)+126(126)=0(0), ZF=1, SF=0, CF=1, OF=0

127(127)+127(127)=254(-2), ZF=0,SF=1,CF=0,OF=1

Сдвиги

- Арифметические сдвиги влево/вправо

```
sal   %eax          // %eax <<= 1
sal   $2, %eax      // %eax <<= 2
sal   %cl, %eax     // %eax <<= %cl & 0x1F
sar   %eax          // %eax >>= 1
sar   $5, %eax      // ...
sar   %cl, %eax     // ...
```

- Логические сдвиги влево/вправо

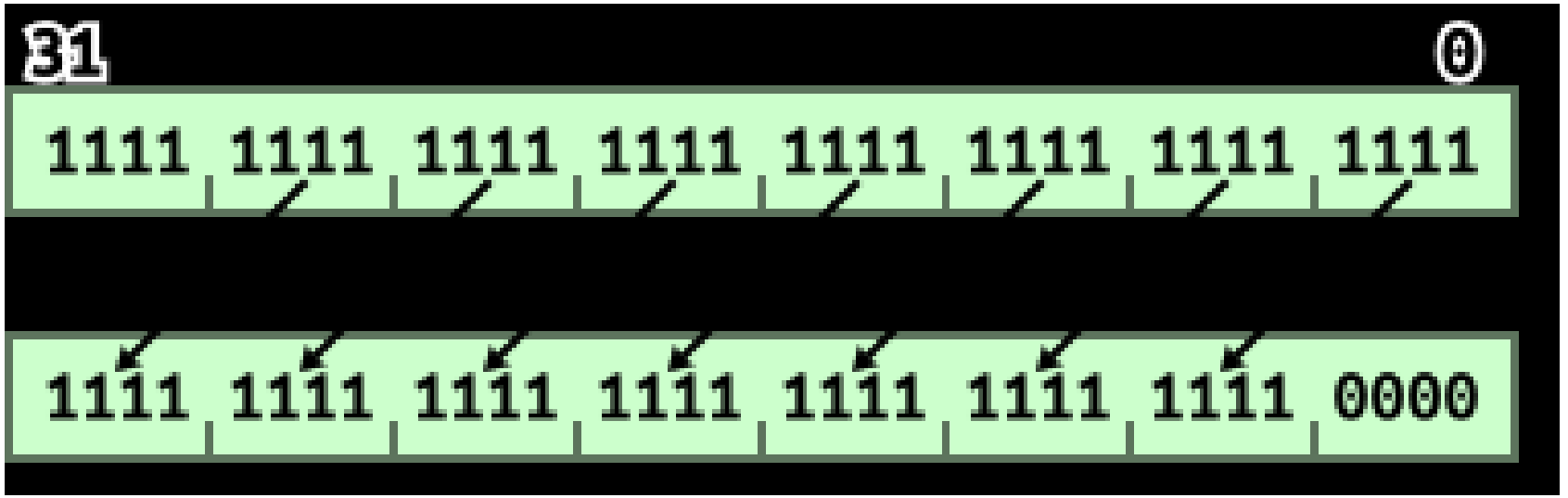
```
shl   [CNT, ] DST // сдвиг влево
shr   [CNT, ] DST // сдвиг вправо
```

Вращения

- Вращение влево/вправо
  rol [CNT,] DST
  ror [CNT,] DST
- Вращение через CF влево/вправо
  rcl [CNT,] DST
  rcr [CNT,] DST

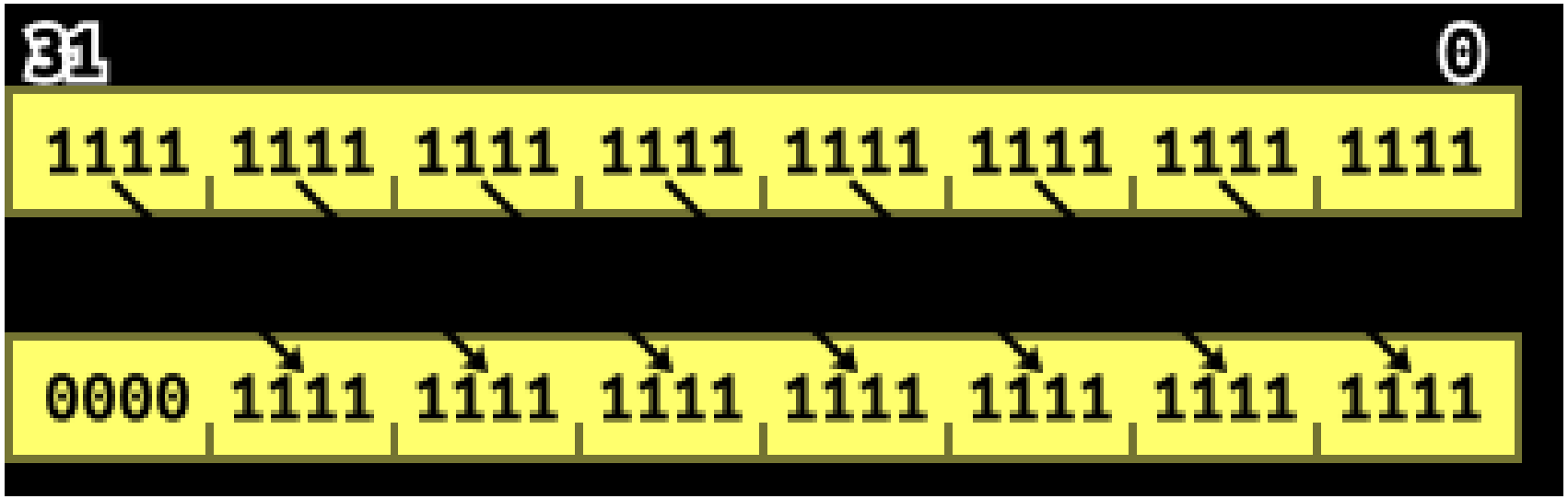
asl/lsl

- asll $4, %eax



lsr

- LSRL $4, %eax

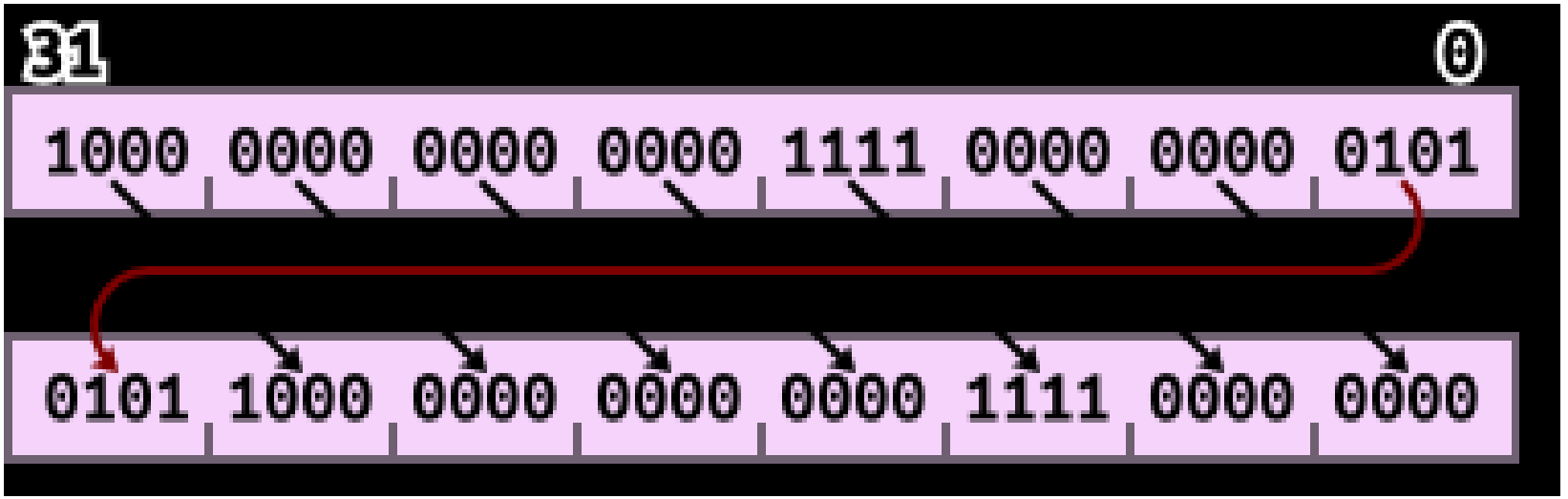


Arithmetical Shift Right

- ASRL $4, %eax

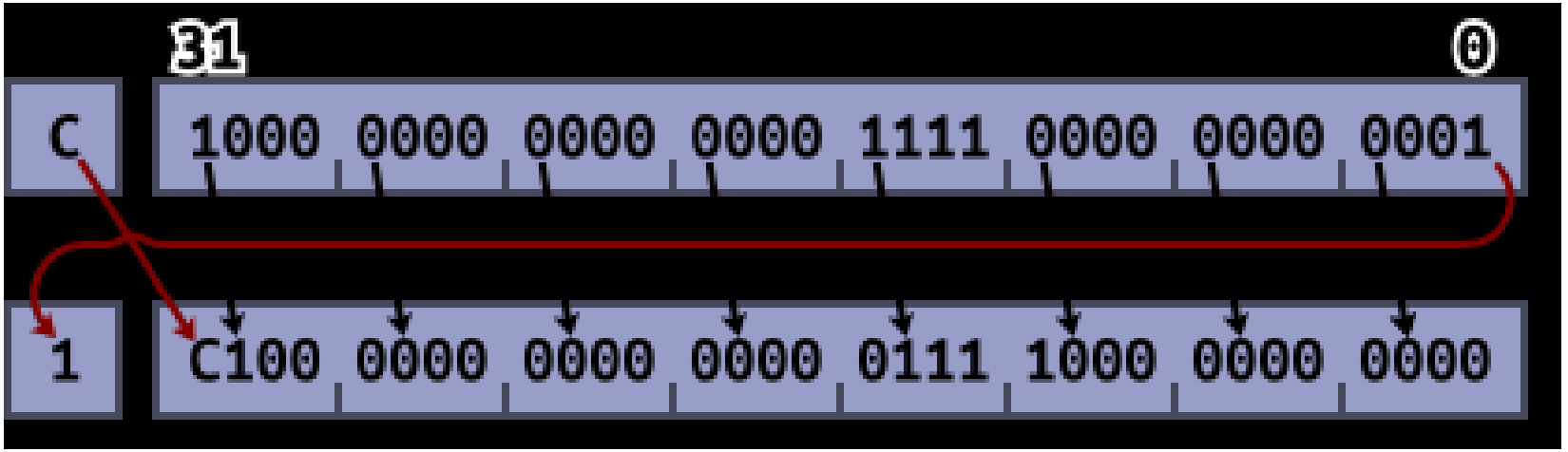
ror

- RORL $4, %eax



rcr

- RCRL %eax


31
0
C
1000 0000 0000 0000 1111 0000 0000 0001
1
C100 0000 0000 0000 0111 1000 0000 0000

Условные переходы

- Переход на метку выполняется только если установлена соответствующая комбинация флагов результата
- Условные переходы по равенству/неравенству
  JE / JZ переход если == или 0
  JNE / JNZ переход если != или не 0

Условные переходы

- Для операций с беззнаковыми числами
  JA / JNBE переход если >
  JAE / JNB / JNC переход если >=
  JB / JNAE / JC переход если <
  JBE / JNA переход если <=
- Для операция со знаковыми числами
  JG / JNLE переход если >
  JGE / JNL переход если >=
  JL / JNGE переход если <
  JLE / JNG переход если <=

Условные переходы

- Специальные случаи

JO	переход если OF == 1
JNO	переход если OF == 0
JS	переход если SF == 1
JNS	переход если SF == 0