

Лекция 11

Язык ассемблера

Структура единицы трансляции

- Программа состоит из секций – логических частей программы
- Компоновщик объединяет содержимое секций из входных объектных файлов, размещает секции в исполняемом файле
- Стандартные секции (минимальный набор)
 - .text – код программы и read-only data
 - .data – глобальные переменные
 - .bss – глобальные переменные, инициализированные нулем

Дополнительные секции

- Можно определять секции с произвольными именами
- Стандартные дополнительные секции:
 - `.rodata`
`.section .rodata, "a"`
- Нестандартные секции:
 - `.string` для размещения строк:
`.section .string, "aMS", @progbits, 1`

Правила использования секций

- Программный код должен размещаться в секции `.text`
- Константы и константные строки могут размещаться в `.text` или в `.rodata`
- Глобальные переменные размещаются в `.data` или `.bss`

Метки (labels, symbols)

- Метки – это символические константы, значение которых известно при компиляции или компоновке программы
 - Метка как адрес, по которому размещается инструкция при выполнении программы
 - Метка как константное значение
- По умолчанию метки видны только в текущей единице компиляции (в том числе объявленные после использования)
- Чтобы сделать метку доступной компоновщику используется `.global NAME`

Точка входа в программу

- Программа должна иметь точку входа – метку, на которую передается управление в начале выполнения программы
- Если компилируем без стандартной библиотеки (-nostdlib), точка входа должна называться `_start` и должна экспортироваться (`.global _start`)
- Если компилируем со стандартной библиотекой, точка входа называется `main` и должна экспортироваться (`.global main`)

Определение данных

- Глобальные переменные:
 - .byte 1, 2, 3, '\n'
 - .short 10, 11
 - .int 0xff00ff00
 - .quad -1
 - .float 1.5
 - .double 2.0
- Строки
 - .ascii "abc"
 - .asciz "Hello" // добавляется неявный \0
- Резервирование памяти под массив
 - .skip 4 * 1024, 0

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

биты:	31	16	15	8	7	0	
				AH	AL		EAX
				BH	BL		EBX
				CH	CL		ECX
				DH	DL		EDX
				SI		ESI	
				DI		EDI	
				BP		EBP	
				SP		ESP	

Специфика РОН

- `%eax` – 32-битный “аккумулятор”
- `%eax:%edx` – пара регистров как 64-битный “аккумулятор”
- `%ecx` – счетчик сдвига
-

Управляющие регистры

- %eip – (instruction pointer) – адрес инструкции, следующей за текущей
- %eflags – регистр флагов процессора
- %cr0 ... %cr4 - прочие управляющие регистры
- %dr0 ... %dr4 – отладочные регистры
- %cs, %ss, %ds, %es, %fs, %gs – “сегментные” регистры

Floating-point registers

- %st(0) ... %st(7) – регистры FPU – каждый имеет размер 80 бит – в настоящее время deprecated
- %mm0 ... %mm7 – регистры MMX (deprecated)
- %xmm0 ... %xmm7 – регистры SSE
- %ymm0 ..., %zmm0 ... - AVX
- SIMD – single instruction multiple data – за одну инструкцию обрабатывается несколько значений

Инструкция mov

- Пересылка данных
movSFX SRC, DST
- Куда пересылаем – второй аргумент!
- SFX – размер пересылаемых данных:
 - 'movb – байт
 - 'movw – слово (16 бит)
 - 'movl – двойное слово (32 бит)
 - 'movq – 64 бит
- Типы пересылок:
 - Регистр-регистр
 - Регистр-память
 - Память-регистр

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

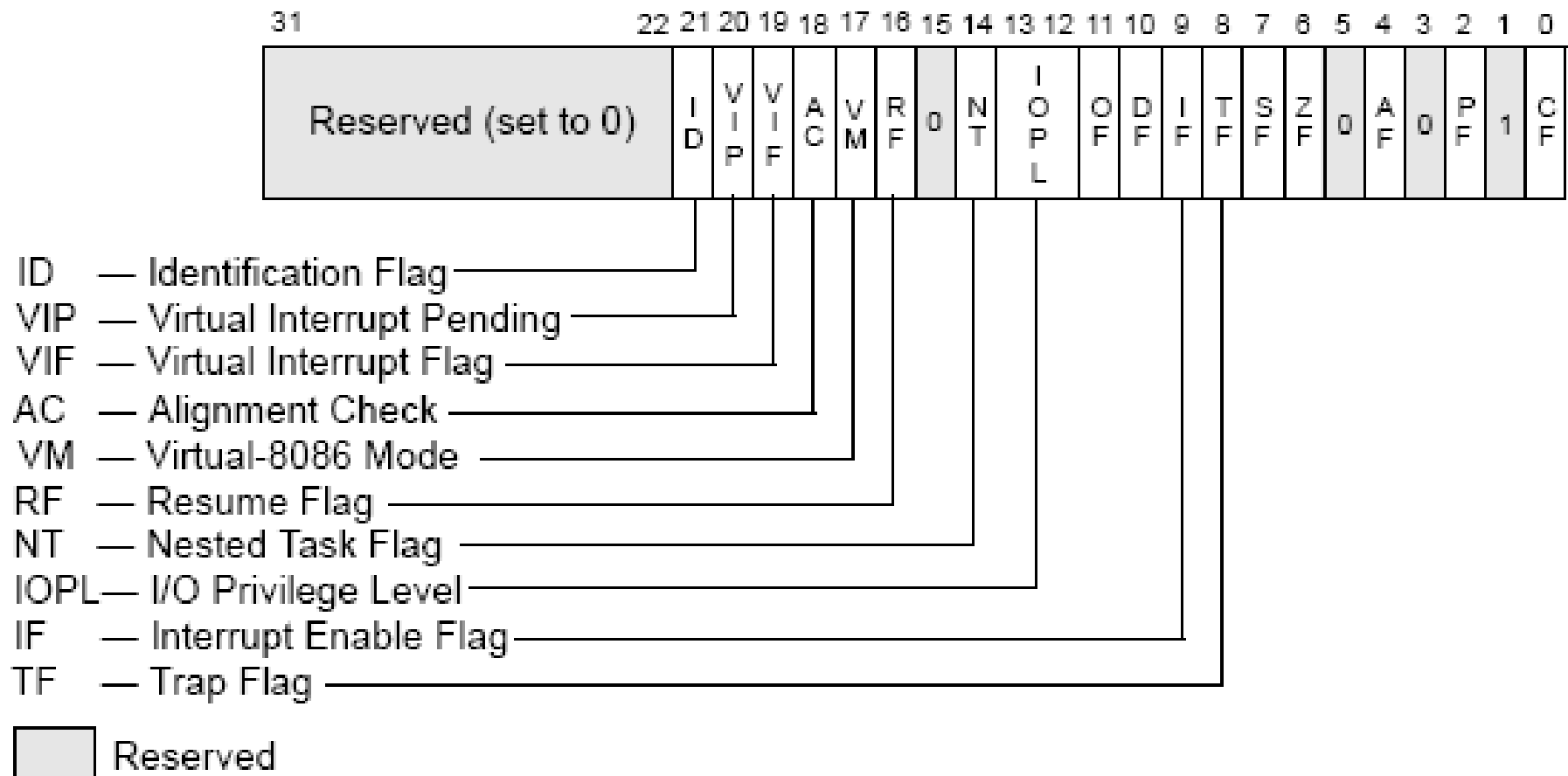
- Арифметические инструкции:
add SRC, DST // DST += SRC
sub SRC, DST // DST -= SRC
cmp SRC1, SRC2 // SRC2 – SRC1
and SRC, DST // DST &= SRC
or SRC, DST // DST |= SRC
xor SRC, DST // DST ^= SRC
test SRC1, SRC2 // SRC1 & SRC2
not DST // DST = ~DST
neg DST // DST = -DST
inc DST // ++DST
dec DST // --DST

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

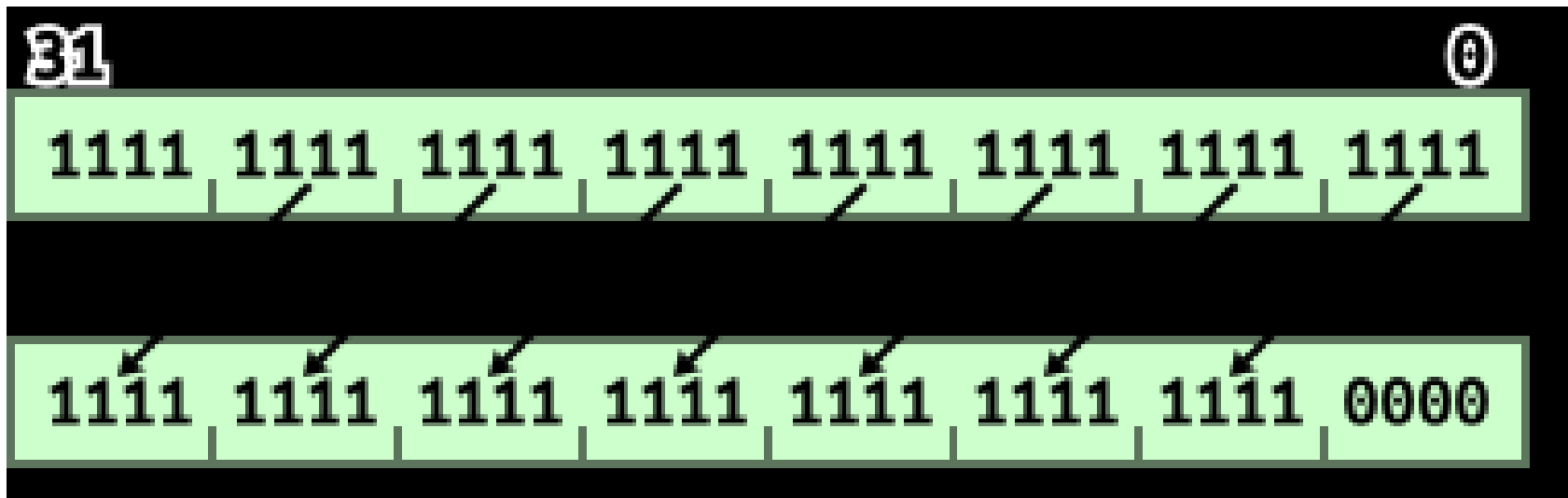
- Арифметические сдвиги влево/вправо
 - sal %eax // %eax <<= 1
 - sal \$2, %eax // %eax <<= 2
 - sal %cl, %eax // %eax <<= %cl & 0x1F
 - sar %eax // %eax >>= 1
 - sar \$5, %eax // ...
 - sar %cl, %eax // ...
- Логические сдвиги влево/вправо
 - shl [CNT,] DST // сдвиг влево
 - shr [CNT,] DST // сдвиг вправо

Вращения

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

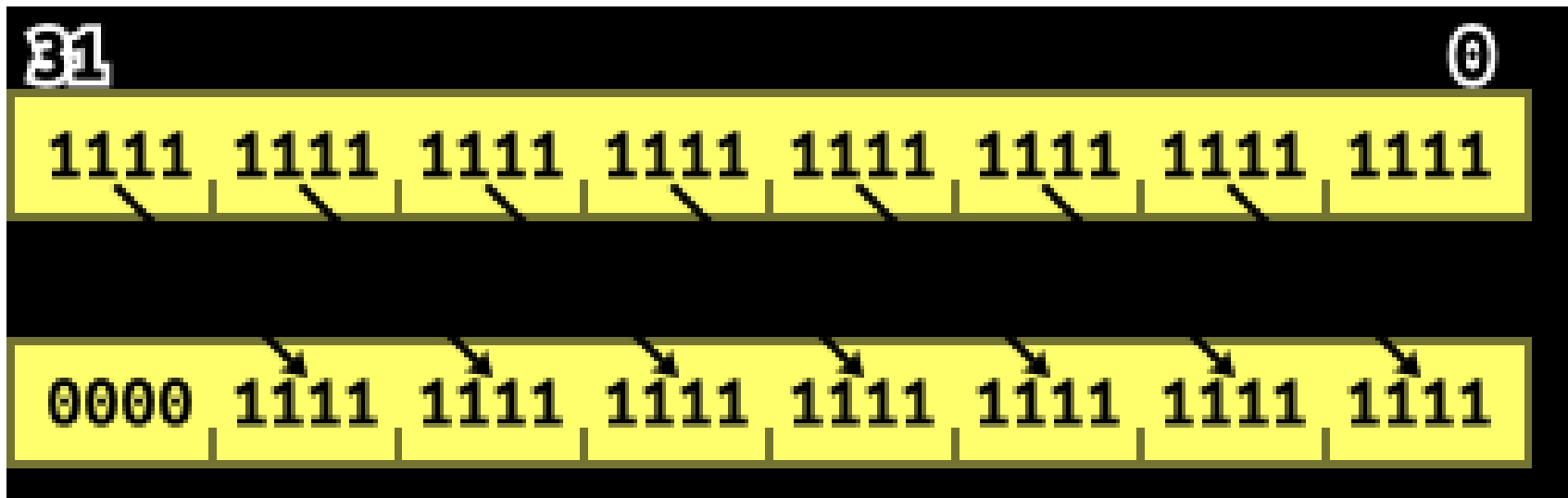
asl/lsl

- `asll $4, %eax`



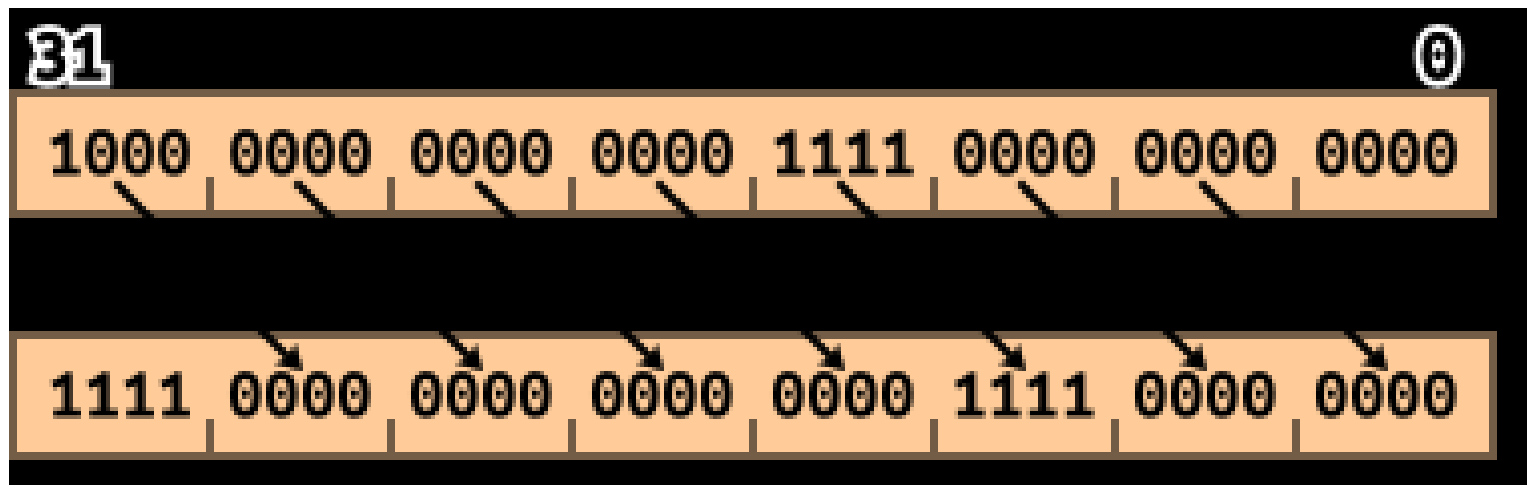
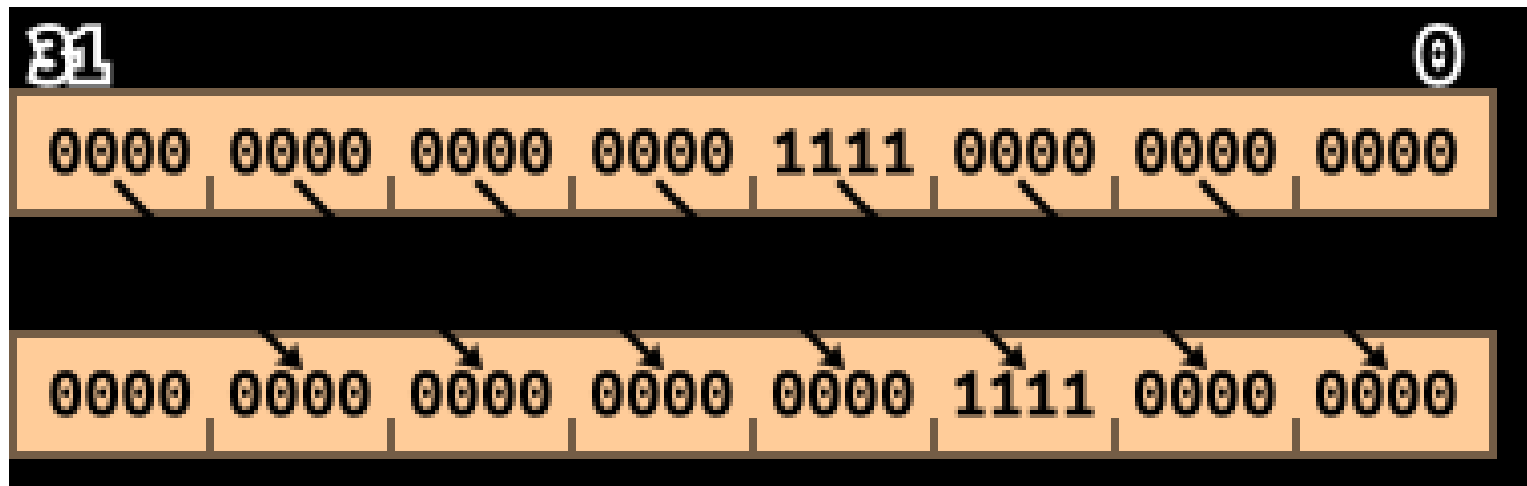
lsr

- LSRL \$4, %eax



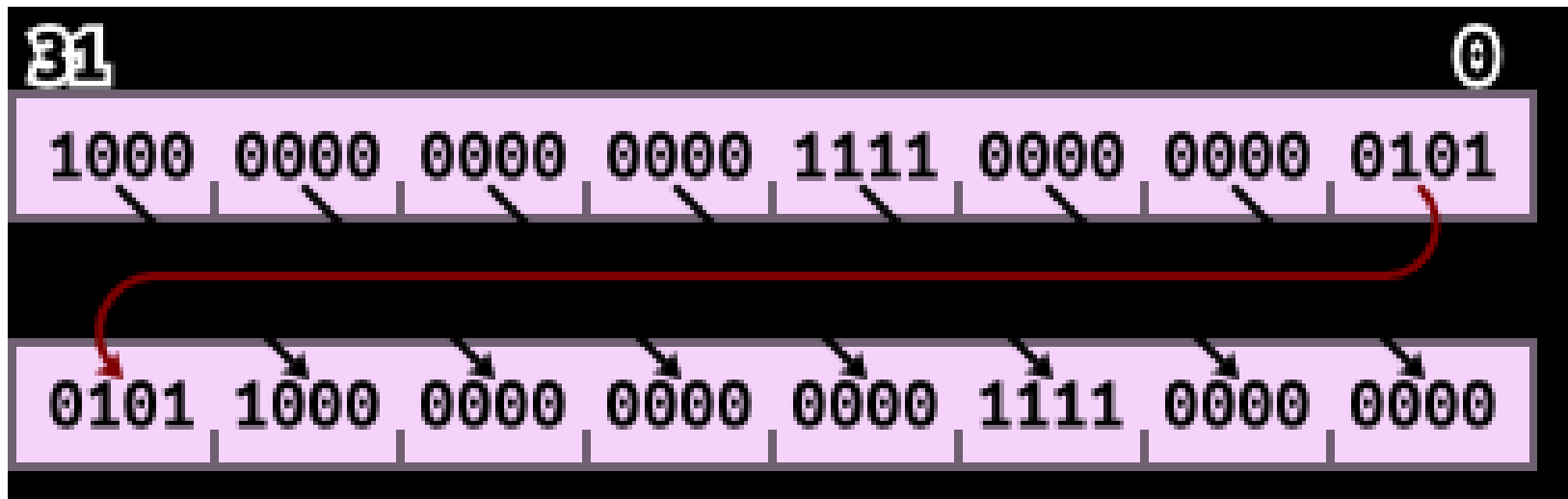
Arithmetical Shift Right

- ASRL \$4, %eax



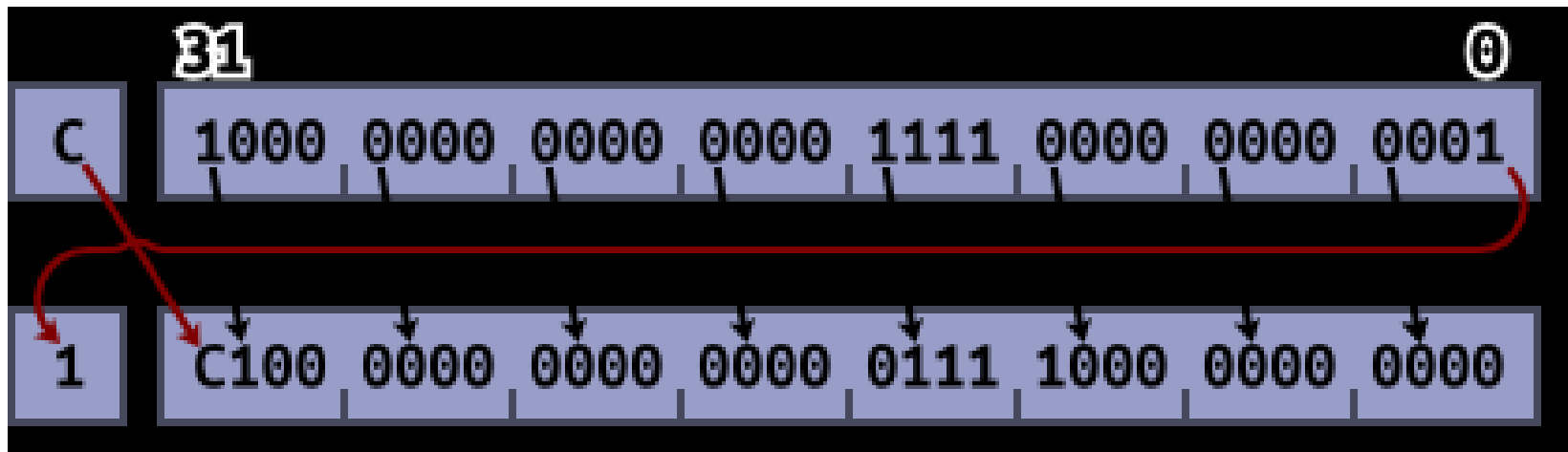
ror

- RORL \$4, %eax



rcr

- RCRL %eax



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 $SF == 0$