КОНСТРУИРОВАНИЕ ПРОГРАММ

Лекция № 09 базовые инструкции IA32

+375 17 293 8039 (505a-5)

+375 17 320 7402 (ОИПИ НАНБ)

prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by/

Кафедра ЭВМ, 2022

2022.04.01

Оглавление

Базовые инструкции IA32

- инструкции перемещения данных;
- арифметические инструкции (двоичная и двоично-десятичная арифметика);
- логические инструкции;
- инструкции сдвигов;
- инструкции для работы с битами и байтами;
- инструкции для работы со строками;
- инструкции передачи управления;
- инструкции ввода/вывода;
- инструкции управления флагами;
- инструкции для работы с сегментными регистрами;
- другие инструкции.

Двоичные арифметические инструкции

Двоичные арифметические команды выполняют базовые вычисления над двоичными целыми числами в байтах, словах и двойных словах, расположенных в памяти и/или регистрах общего назначения.

ADD — сложение целых;
SUB — вычитание;
IMUL — умножение целых со знаком;
MUL — умножение беззнаковых целых;
IDIV — деление целых со знаком;
DIV — деление беззнаковых целых;
ADC — сложение целых с переносом;
SBB — вычитание с заемом (отрицательным переносом);
ADCX — сложение беззнаковых целых с переносом;
ADOX — сложение беззнаковых целых с переполнением;
INC — инкремент;
DEC — декремент;
NEG — сменить знак;
CMP — арифметическое сравнение.

ADC — сложение целых с переносом

```
ADC     AL,     imm8
ADC     AX,     imm16
ADC     EAX,    imm32

ADC     r/m8,   imm8
ADC     r/m16,  imm8
ADC     r/m32,  imm8
ADC     r/m16,  imm16
ADC     r/m32,  imm32
ADC     r/m,    r           ; 8, 16, 32
ADC     r,      r/m         ; 8, 16, 32
```

Операция

```
DEST ← (DEST + SRC + CF)
```

Описание

Выполняет сложение операнда-назначения (первый операнд), операнда-источника (второй операнд), флаг переноса (**CF**) и сохраняет результат в операнде-назначении.

Операндом-назначением может быть регистр или ячейка памяти.

Операндом-источником может быть непосредственное значение, регистр или ячейкой памяти. (Однако два операнда памяти не могут использоваться в одной инструкции.)

Состояние флага **CF** представляет перенос при выполнении предыдущей инструкции сложения.

В том случае, когда в качестве операнда-источника используется непосредственное значение, оно расширяется до длины формата операнда-назначения.

Инструкция ADC не различает операнды со знаком и без знака.

Вместо этого процессор вычисляет результат для обоих типов данных и устанавливает флаги **OF** и **CF**, указывая наличие переноса в обоих случаях.

Флаг **SF** указывает знак результата для знакового сложения.

Инструкция **ADC** обычно выполняется как часть многобайтового или многословного сложения, в котором инструкция ADC следует за инструкцией **ADD**.

Данная инструкция может использоваться с префиксом **LOCK**, чтобы позволить ей выполняться атомарно.

Флаги OF, SF, ZF, AF, CF, PF устанавливаются в соответствии с результатом.

SBB — вычитание с заемом (borrow)

```
SBB     AL, imm8
SBB     AX, imm16
SBB     EAX, imm32

SBB     r/m8, imm8
SBB     r/m16, imm8
SBB     r/m32, imm8
SBB     r/m16, imm16
SBB     r/m32, imm32
SBB     r/m, r       ; 8, 16, 32
SBB     r, r/m       ; 8, 16, 32
```

Операция

```
DEST ← (DEST – (SRC + CF))
```

Описание

Выполняет сложение операнда-источника (второй операнд) и флага переноса (**CF**) и вычитает результат из операнда-назначения (первый операнд). Результат вычитания сохраняется в операнде-назначении.

Операндом-назначением может быть регистр или ячейка памяти.

Операндом-источником может быть непосредственное значение, регистр или ячейкой памяти. (Однако два операнда памяти не могут использоваться в одной инструкции.)

Состояние флага **CF** представляет заем (отрицательный перенос), случившийся при выполнении предыдущей инструкции вычитания.

В том случае, когда в качестве операнда-источника используется непосредственное значение, оно расширяется до длины формата операнда-назначения.

Инструкция SBB не различает операнды со знаком и без знака.

Вместо этого процессор вычисляет результат для обоих типов данных и устанавливает флаги **OF** и **CF**, указывая наличие переноса в обоих случаях.

Флаг **SF** указывает знак результата для знакового сложения.

Инструкция **SBB** обычно выполняется как часть многобайтового или многословного вычитания, в котором инструкция **SBB** следует за инструкцией **SUB**.

Данная инструкция может использоваться с префиксом LOCK, чтобы позволить ей выполняться атомарно.

Флаги **OF**, **SF**, **ZF**, **AF**, **CF**, **PF** устанавливаются в соответствии с результатом.

Intel ADX — Расширение инструкций сложения с переносом (многократной точности)

Обе инструкции являются более эффективными вариантами существующей инструкции ADC, с той разницей, что каждая из двух новых инструкций влияет только на один флаг, в то время как ADC как дополнение со знаком может устанавливать как флаги переполнения, так и флаги переноса, а также сбросить остальные флаги процессора.

Наличие двух версий означает, что две цепочки сложений многократной сложности могут вычисляться параллельно, не мешая друг другу.

AMD добавила поддержку этих инструкций, начиная с Ryzen.

Инструкция	Описание
ADCX	Складывает два целых числа без знака плюс перенос, считывая перенос из флага переноса CF и, при необходимости, там же его и устанавливает. Не влияет на другие флаги, кроме флага переноса.
ADOX	Складывает два целых числа без знака плюс перенос, считывая перенос из флага переполнения OF и, при необходимости, там же его и устанавливает. Не влияет на другие флаги, кроме флага переполнения.

ADCX — беззнаковое сложение двух операндов с переносом

```
ADCX    r32,r/m32
```

Операция

```
DEST ← (DEST + SRC + CF)
```

Выполняет беззнаковое сложение операнда-назначения (первого операнда), операнда-источника (второго операнда), флага переноса (**CF**) и сохраняет результат в операнде-назначении.

Операндом-назначеним является регистр общего назначения.

Операндом-источником может быть регистр общего назначения или ячейка памяти.

Состояние **CF** может представлять собой перенос из предыдущего сложения. Инструкция устанавливает флаг **CF** в перенос, генерируемый беззнаковым сложением операндов.

Инструкция **ADCX** предназначена для выполнения в контексте сложения с многократной точностью, где выполняется серия последовательных сложений нескольких операндов с учетом переносов. В начале серии необходимо убедиться, что **CF** находится в желаемом начальном состоянии.

Обычно начальное состояние **CF** должно быть 0, что достигается с помощью какой-нибудь инструкции, очищающей **CF** (например, **XOR**).

Инструкция поддерживается в реальном режиме и в режиме виртуального 8086. Размер операнда всегда составляет 32 бита.

Примечание: **ADCX** определяет флаг **OF** иначе, чем инструкции **ADD/ADC**.

Флаг **CF** устанавливается/сбрасывается по результату.

Флаги **OF**, **SF**, **ZF**, **AF**, **PF** остаются без изменения.

ADOX — сложение беззнаковых целых с флагом переполнения

```
ADOX    r32, r/m32
```

Операция

```
DEST ← (DEST + SRC + OF)
```

Описание

Выполняет беззнаковое сложение операнда-назначения (первого операнда), операнда-источника (второго операнда), флага переполнения (**OF**) и сохраняет результат в операнде-назначении.

Операндом-назначением является регистр общего назначения.

Операндом-источником может быть регистр общего назначения или ячейка памяти.

Состояние **OF** может представлять собой переполнение из предыдущего сложения.

Инструкция устанавливает флаг **OF** в перенос, генерируемый беззнаковым сложением операндов.

Инструкция **ADOX** предназначена для выполнения в контексте сложения с многократной точностью, где выполняется серия последовательных сложений нескольких операндов с учетом переносов. В начале серии необходимо обнулить **OF** (например, **XOR**).

Флаг **OF** устанавливается/сбрасывается по результату.

Флаги **CF**, **SF**, **ZF**, **AF**, **PF** остаются без изменения.

Пример сложения произвольной точности

```
global  _add128

section .data
                            ; 92254330539917530352666022345239189112
a128            dd   12345678h, 23456789h, 3456789ah, 456789abh
                            ; 24197857166293592664352578415092009387
b128            dd   456789abh, 3456789ah, 23456789h, 12345678h

section .bss
sum             resd 4          ; 116452187706211123017018600760331198499

section .text
_add128:
                mov     ecx, 4              ; двойных слов в операнде
                xor     esi, esi            ; очистим индекс, CF и OF (xor)
.@:
                mov     eax, [a128 + 4*esi] ; первый операнд
                mov     ebx, [b128 + 4*esi] ; второй операнд
                adc     eax, ebx            ; возможно установился CF
;               adcx    eax, ebx            ; CPUID:ADX
                mov     [sum + 4*esi], eax  ; сохраним частную сумму
                inc     esi
                loop    .@
fin:
                mov     eax, 1              ; системный вызов exit()
                mov     ebx, 0
                int     0x80
```

INC/DEC — инкремент/декремент на единицу

```
INC/DEC    r/m  ;  8, 16, 32
INC/DEC    r    ; 16, 32
```

Операция

```
DEST ← DEST + 1
DEST ← DEST - 1
```

Описание

INC добавляет 1 к операнду-приемнику, а DEC вычитает 1 из операнда-приемника, сохраняя при этом состояние флага **CF**. Операндом-приемником может быть регистр или ячейка памяти.

Эти инструкции позволяют обновлять счетчик цикла без нарушения флага CF.

Для выполнения операции приращения/уменьшения, которые обновляют флаг **CF**, следует использовать инструкции **ADD/SUB**, соответственно, с непосредственным операндом 1.

Данные инструкции могут использоваться с префиксом **LOCK**, чтобы обеспечить их атомарное выполнение.

Флаг **CF** остается без изменения.

Флаги **OF**, **SF**, **ZF**, **AF**, **PF** устанавливаются в соответствии с результатом операции.

NEG — сменить знак

```
NEG    r/m    ; 8, 16, 32
```

Описание

Заменяет значение операнда (операнд-приемник) дополнением до двух. (Эта операция эквивалентна вычитанию операнда из 0.)

Операнд-приемник находится в регистре общего назначения или в ячейке памяти.

Данная инструкция может использоваться с префиксом **LOCK**, чтобы обеспечить атомарное выполнение инструкции.

Флаги

Если операнд-источник равен 0, флаг **CF** сбрасывается.

В противном случае **CF** устанавливается.

Флаги **OF**, **SF**, **ZF**, **AF**, **PF** устанавливаются в соответствии с результатом операции.

CMP — Сравнить два операнда

```
CMP     AL, imm8
CMP     AX, imm16
CMP     EAX, imm32

CMP     r/m, imm      ; 8, 16, 32
CMP     r/m16, imm8
CMP     r/m32, imm8

CMP     r/m, r        ; 8, 16, 32
CMP     r, r/m        ; 8, 16, 32
```

Сравнивает первый операнд-источник со вторым операндом-источником и устанавливает флаги состояния в регистре **EFLAGS** в соответствии с результатами.

Сравнение выполняется путем вычитания второго операнда из первого операнда, после чего флаги состояния устанавливаются таким же образом, как в случае инструкции **SUB**.

В случае, когда в качестве операнда используется непосредственное значение (**imm**), оно расширяется со знаком до длины первого операнда (**imm** может быть только вторым операндом).

Коды условий, используемые командами **Jcc**, **CMOVcc** и **SETcc**, основываются на результатах команды **CMP**.

Флаги **OF**, **CF**, **SF**, **ZF**, **AF**, **PF** устанавливаются в соответствии с результатом операции.

Десятичные арифметические инструкции

Десятичные арифметические инструкции выполняют десятичные вычисления над двоично-десятичными (BCD) данными.

DAA — коррекция регистра AL после сложения упакованных BCD
DAS — коррекция регистра AL после вычитания упакованных BCD

AAA — коррекция регистра AL после сложения неупакованных BCD
AAS — коррекция регистра AL после вычитания неупакованных BCD

AAM — коррекция регистра AX после умножения
AAD — коррекция регистра AX перед делением

Отдельных команд для выполнения арифметических операций над BCD-числами нет — используются те, что предназначены для работы с двоичными представлениями.

Поэтому после сложения, вычитания и умножения, а также перед делением необходимо BCD-представления преобразовать в такое, чтобы результат двоичной операции был действительным BCD-числом.

Десятичная арифметика выполняется с помощью объединения двоичных арифметических инструкций **ADD**, **SUB**, **MUL** и **DIV** с десятичными арифметическими инструкциями.

Десятичные арифметические инструкции предоставляются для выполнения следующих операций:

- скорректировать результаты предыдущей двоичной арифметической операции для получения правильного результата в представлении BCD.
- скорректировать операнды последующей двоичной арифметической операции, чтобы эта операция давала действительный результат в представлении BCD.

Данные инструкции используются как для упакованных, так и для распакованных BCD.

Десятичные арифметические инструкции делятся на подгруппы инструкций:

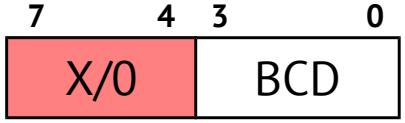
- коррекция упакованных BCD-чисел;
- коррекция распакованных BCD-чисел.

Распакованные BCD (Binary Coded Decimal) и упакованные BCD целые числа

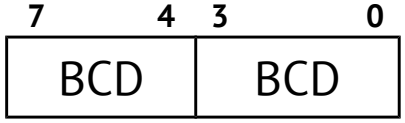
Десятичные целые числа в двоичном коде — это 4-битные целые числа без знака с допустимыми значениями в диапазоне от 0 до 9.

Для выполнения операций над целыми BCD числами они должны располагаться либо в одном или нескольких регистрах общего назначения, либо в одном или нескольких регистрах x87 FPU.

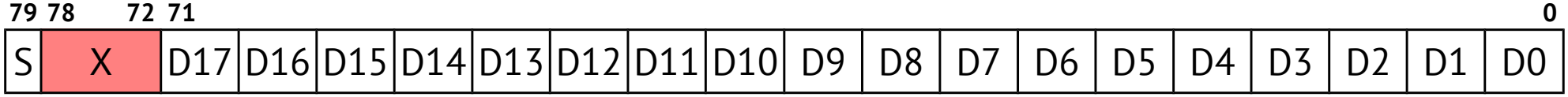
Распакованные целые BCD — одно BCD в байте (X должно быть 0 в случае умножения/деления).



Упакованные BCD целые — два BCD в байте.



Упакованные BCD целые размером 80 бит (в FPU они преобразуются в IEEE754.64)



DAA/DAS — десятичная коррекция регистра AL после сложения/вычитания

```
DAA     ; последовательность инструкций -- (ADD, DAA)
DAS     ; последовательность инструкций -- (SUB, DAS)
```

Описание

Корректируют сумму/разность двух упакованных BCD, чтобы в результате получилось упакованное BCD.

Инструкции операндов не имеют, но используют регистр **AL** как в качестве источника, так и в качестве приемника.

Эти инструкции нужна только тогда, когда они следуют за инструкциями **ADD/SUB**, соответственно, которые выполняют сложение/вычитание (двоичное) двух двухзначных упакованных BCD чисел и сохраняют байт результата в регистре **AL**.

После этого следует вызвать, соответственно, инструкцию **DAA/DAS**, которая скорректирует содержимое регистра **AL** так, чтобы он содержал правильный двухзначный результат в формате упакованного **BCD**.

Если в результате сложения происходит десятичный перенос/заем, устанавливаются флаги **CF** и **AF**. Если десятичного переноса/заема не было, флаги **CF** и **AF** очищаются.

Эта инструкция выполняется, как описано в режиме совместимости и традиционном 32-разрядном режиме. Недопустима в 64-битном режиме.

Флаги AF и CF устанавливаются если коррекция приводит к десятичному переносу/ заему в любой цифре результата.

Флаги SF, ZF, PF устанавливаются в соответствии с результатом.

Флаг OF не определен.

AAA/AAS — ASCII коррекция регистра AL после сложения/вычитания

```
AAA     ; последовательность инструкций -- (ADD, AAA)
AAS     ; последовательность инструкций -- (SUB, AAS)
```

Корректируют сумму/разность двух распакованных BCD, чтобы в результате получилось распакованное BCD.

Инструкции операндов не имеют, но используют регистр **AL** как в качестве источника, так назначения.

Эта инструкция нужна только тогда, когда она следует за инструкцией **ADD/SUB**, которая выполняет сложение/ вычитание (двоичное) двух распакованных BCD и сохраняет байт результата в регистре **AL**. После этого должна быть вызвана инструкция **AAA/AAS**, которая скорректирует содержимое регистра **AL** так, чтобы он содержал правильную цифру результата в формате распакованного **BCD**.

Если в результате сложения/вычитания происходит десятичный перенос/заем, регистр **AH** увеличивается/уменьшается на 1 и устанавливаются флаги **CF** и **AF**.

Если десятичного переноса/заема не было, флаги **CF** и **AF** очищаются, а регистр **AH** не изменяется. В любом случае биты с 4 по 7 регистра **AL** устанавливаются в 0.

Эта инструкция выполняется, как описано в режиме совместимости и в традиционном 32-разрядном режиме. Недопустима в 64-битном режиме.

Флаги AF и CF устанавливаются или очищаются в соответствии с результатом.

Флаги OF, SF, ZF, PF не определены.

AAM — ASCII коррекция регистра AX после умножения

```
AAM          ; последовательность инструкций -- (MUL, AAM)
AAM    imm8  ; коррекция по основанию счисления != 10
```

Описание

Корректирует результат умножения двух распакованных BCD с целью получения пары распакованных BCD (по основанию 10).

Инструкция операндов не имеет, но использует регистр **AX** как в качестве источника, так назначения.

Эта инструкция нужна только тогда, когда она следует за инструкцией **MUL**, которая выполняет умножение (двоичное) двух распакованных BCD и сохраняет слово результата в регистре **AX**. После этого должна быть вызвана инструкция **AAM**, которая скорректирует содержимое регистра **AX** так, чтобы он содержал правильный 2-х значный результат (по основанию 10) в формате распакованного **BCD**.

Обобщенная версия данной инструкции позволяет корректировать содержимое **AX** для создания пары распакованных BCD цифр по любому основанию.

Мнемоника **AAM** интерпретируется всеми ассемблерами как означающая корректировку значения по основанию 10. Чтобы скорректировать **AX** по другому основанию, инструкция должна быть вручную закодирована в машинном коде (**D4 imm8**).

Эта инструкция недопустима в 64-битном режиме.

Флаги SF, ZF, and PF устанавливаются или очищаются в соответствии с результатом, полученным в регистре AL.

Флаги OF, AF, CF неопределены.

AAD — ASCII коррекция регистра AX перед делением

```
AAD           ; последовательность инструкций -- (AAD, DIV)
AAD    imm8   ; коррекция по основанию счисления
```

Описание

Выполняет коррекцию двух распакованных BCD цифр, располагающихся в регистрах **AL** (младшая значащая) и **AH** (старшая значащая) так, чтобы операция деления, выполненная для результата (**AX**), давала правильное значение распакованного BCD числа.

Эта инструкция нужна только тогда, когда она предшествует инструкции **DIV**, которая делит в двоичном режиме скорректированное значение регистра AX на распакованное BCD число.

Инструкция **AAD** устанавливает значение в регистре **AL** в **(AL + (10 * AH))**, после чего очищает регистр **AH** в **00H**. В результате значение в регистре **AX** становится равным двоичному эквиваленту исходного распакованного двузначного числа (по основанию 10), расположенного в регистрах **AH** и **AL**.

Обобщенная версия данной инструкции позволяет корректировать две распакованные цифры по любому основанию, расположенному в байте **imm8** (например, **08H** для восьмеричной, **0AH** для десятичной или **0CH** для основания, равного 12).

Мнемоника **AAD** интерпретируется всеми ассемблерами как означающая корректировку значений ASCII по основанию 10. Чтобы скорректировать значения **AX** по другому основанию, инструкция должна быть вручную закодирована в машинном коде (**D5 imm8**).

Эта инструкция недопустима в 64-битном режиме.

Флаги SF, ZF, and PF устанавливаются или очищаются в соответствии с результатом, полученным в регистре AL. Флаги OF, AF, CF неопределены.

Логические инструкции

Логические инструкции выполняют базовые логические операции И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ и НЕ над байтами, словами и двойными словами.

AND — побитовое логическое И;
OR — побитовое логическое ИЛИ;
XOR — побитовое логическое ИСКЛЮЧАЮЩЕЕ ИЛИ;
NOT — побитовое логическое НЕ;

AND/OR/XOR — побитовая логическая операция

```
AND/OR/XOR    AL, imm8       ; AL ← AL (OP) imm8
AND/OR/XOR    AX, imm16      ; AX ← AX (OP) imm16
AND/OR/XOR    EAX, imm32     ; EAX ← EAX (OP) imm32

AND/OR/XOR    r/m,   imm     ; r     ← r (OP) imm (8, 16, 32)
AND/OR/XOR    r/m16, imm8    ; r/m16 ← r/m16 (OP) SignExt(imm8)
AND/OR/XOR    r/m32, imm32   ; r/m32 ← r/m32 (OP) SignExt(imm8)

AND/OR/XOR    r/m, r         ; r/m ← r/m (OP) r (8, 16, 32)
AND/OR/XOR    r, r/m         ; r   ← r (OP) r/m (8, 16, 32)
```

Операция

```
DEST ← DEST (OP) SRC
```

Описание

Выполняет побитовую операцию **(OP)** (одну из **AND**, **OR**, **XOR**) над операндом-приемником (первым) и операндом-источником (вторым), после чего сохраняет результат в местоположение операнда-приемника.

Операндом-источником может быть непосредственное значение, регистр или ячейка памяти.

Операндом-приемником может быть регистр или ячейка памяти. (Однако два операнда памяти не могут использоваться в одной инструкции.)

Каждый бит результата устанавливается согласно таблиц истинности, приведенных ниже.

AND	0	1
0	0	0
1	0	**1**

OR	0	1
0	0	**1**
1	**1**	**1**

XOR	0	1
0	0	**1**
1	**1**	0

XNOR	0	1
	1	0
	0	**1**

Данные инструкции могут использоваться с префиксом **LOCK**, чтобы обеспечить атомарное выполнение.

Флаги

Флаги **OF** и **CF** обнуляются. Флаги **SF**, **ZF** и **PF** устанавливаются в согласии с результатами операции. Состояние флага **AF** неопределено.

NOT — побитовое логическое НЕ (инвертирование бит)

```
NOT    r/m    ; 8, 16, 32
```

Операция

```
DEST ← NOT DEST
```

Описание

Выполняет побитовую операцию **NOT** (для каждого 1 установливает начение 0, а для каждого 0 установливает значение 1) для операнда и сохраняет результат в местоположении операнда-приемника.

Операндом-приемником может быть регистр или ячейка памяти.

Данная инструкция может использоваться с префиксом **LOCK**, чтобы обеспечить атомарное выполнение.

Флаги

Флаги не меняются

Инструкции сдвигов и циклических сдвигов (Shift and Rotate)

Инструкции сдвигов и циклических сдвигов переставляют биты в операнде. Эти инструкции подразделяются на подгруппы:

SAL/SHL/SAR/SHR — инструкции сдвига бит ;

SHLD/SHRD — инструкции двойного сдвига бит между операндами;

ROR/ROL/RCR/RCL — инструкции циклического сдвига бит.

SAL/SHL/SAR/SHR — инструкции сдвига бит

```
Sxy    r/m, 1     ; 8, 16, 32 — сдвиг на один разряд
Sxy    r/m, CL    ; 8, 16, 32 — сдвиг на CL разрядов
Sxy    r/m, imm8  ; 8, 16, 32 — сдвиг на imm8 разрядов
```

x — H (логический), A (арифметический)

y — L (влево), R (вправо)

Инструкции **SAL** (арифметический сдвиг влево), **SHL** (логический сдвиг влево), **SAR** (арифметический сдвиг вправо), **SHR** (логический сдвиг вправо) выполняют арифметическое или логическое смещение битов в байте, слове или двойном слове.

Инструкции сдвигают биты в первом операнде (операнде-приемнике) влево или вправо на количество битов, указанное во втором операнде (операнде-счетчике).

Биты, сдвинутые за пределы границы операнда-приемника, сначала сдвигаются во флаг **CF**, а затем отбрасываются.

В конце операции сдвига флаг CF содержит последний сдвинутый бит из операнда-приемника.

Операндом-приемником может быть регистр или ячейка памяти.

Операндом-счетчиком — непосредственное значение или регистр CL.

Значение операнда-счетчика маскируется 5-ю битами.

Диапазон счетчика ограничен от 0 до 31.

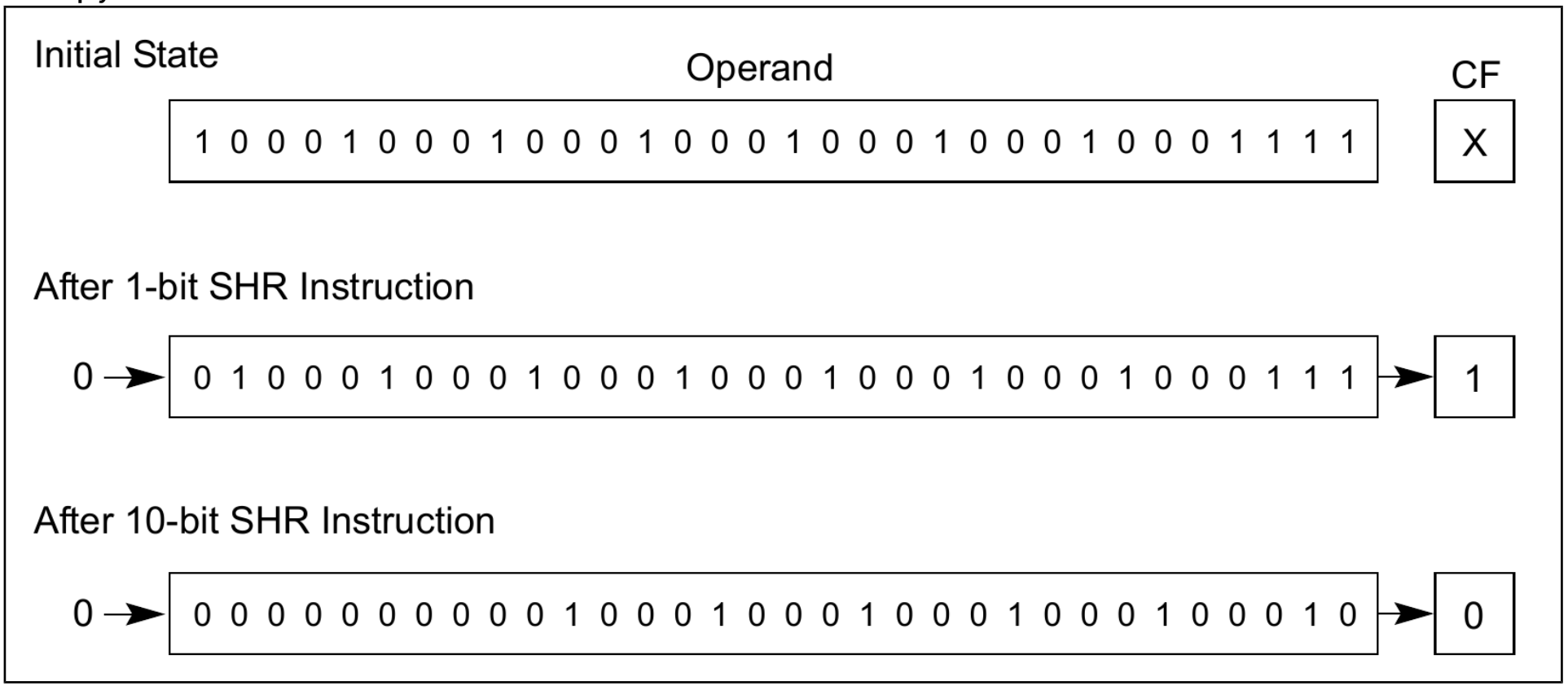
```
1 00000000 D1E0      shl  eax, 1
2 00000002 C1E002    shl  eax, 2
3 00000005 D3E0      shl  eax, cl
```

Команды арифметического сдвига влево (**SAL**) и логического сдвига влево (**SHL**) выполняют одну и ту же операцию — они сдвигают биты в операнде-приемнике влево (к более значимым битовым позициям).

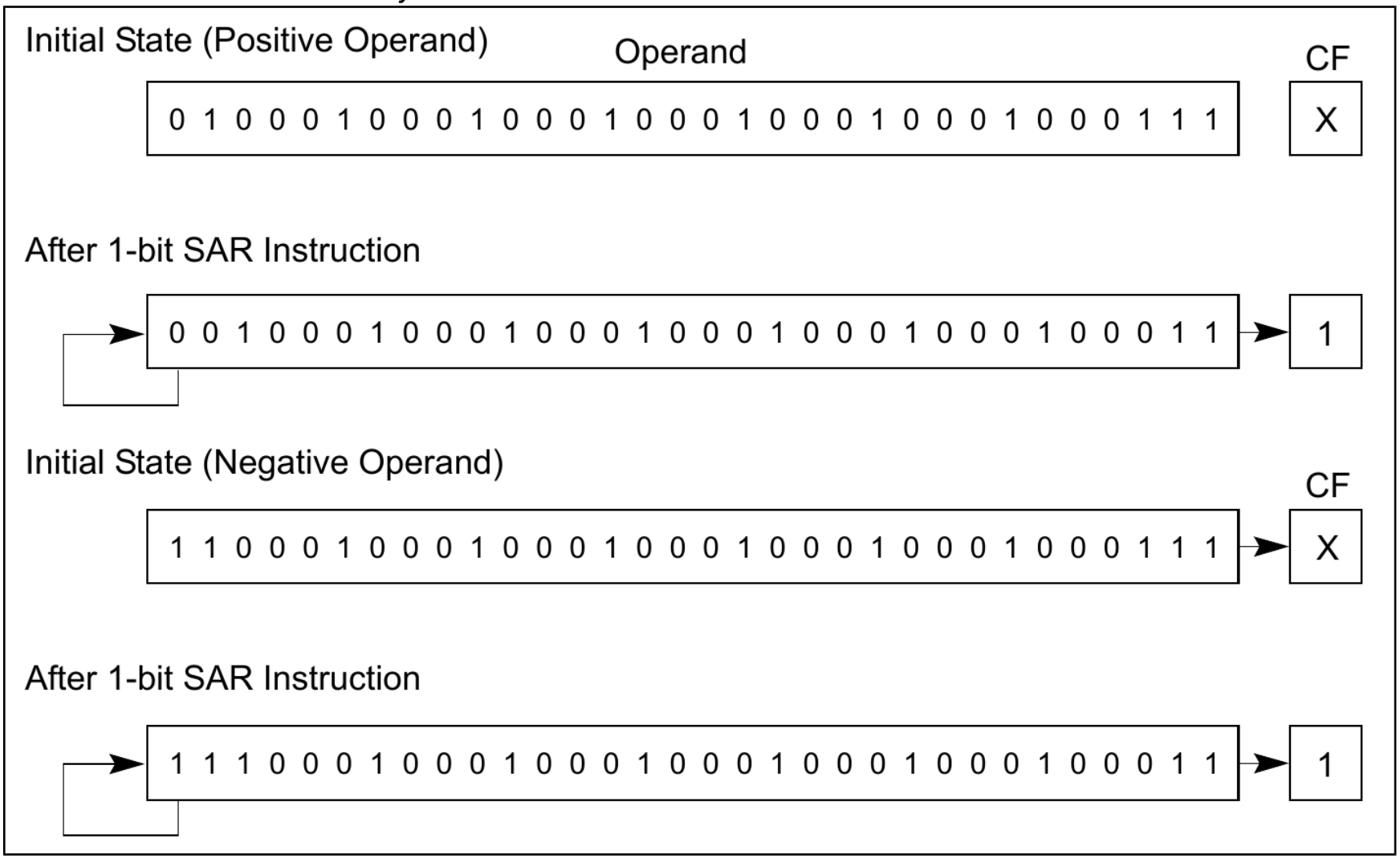
Для каждого значения счетчика сдвига самый старший бит операнда-приемника смещается во флаг **CF**, а младший бит очищается.

Команды арифметического сдвига вправо (**SAR**) и логического сдвига вправо (**SHR**) сдвигают биты операнда-приемника вправо (в направлении менее значимых битовых позиций).

Инструкция **SHR** очищает наиболее значимый бит.



Инструкция **SAR** устанавливает или очищает старший значащий бит в соответствии со знаком (старший значащий бит) исходного значения в операнде-приемнике. По сути, инструкция **SAR** заполняет смещенное значение пустой битовой позиции знаком несмещенного значения.



Для каждого значения счетчика сдвига младший значащий бит операнда-адресата сдвигается во флаг **CF**, а самый старший бит устанавливается или сбрасывается в зависимости от типа команды.

Для значения счетчика, равного 1 (imm8), предусмотрено специальное кодирование кода операции.

Команды **SAR** и **SHR** могут использоваться для выполнения деления со знаком или без знака, соответственно, операнда-адресата на степени двойки. Например, использование команды **SAR** для сдвига целого со знаком на 1 бит вправо делит значение на 2.

Использование инструкции SAR для выполнения операции деления не дает того же результата, что и инструкция IDIV.

Частное от деления с помощью инструкции IDIV округляется в сторону нуля, тогда как «частное» для инструкции SAR округляется в сторону отрицательной бесконечности.

Эта разница имеет место только для отрицательных чисел.

Например, когда инструкция **IDIV** используется для деления -9 на 4, результат равен -2 с остатком -1. Если команда **SAR** используется для сдвига -9 вправо на два бита, результат равен -3, а «остаток» равен +3, при этом инструкция **SAR** сохраняет только самый старший бит остатка (в флаге **CF**).

Совместимость

8086 не маскирует счетчик сдвига. Тем не менее, все остальные процессоры IA-32 (начиная с процессора Intel 286) маскируют счетчик сдвига 5-ю битами, что приводит к максимальному счету 31. Это маскирование выполняется во всех режимах работы (включая режим virtual-8086) с целью уменьшить максимальное время выполнения инструкций.

Флаги

CF

Флаг **CF** содержит значение последнего бита, сдвинутого из операнда-приемника.

Если счетчик сдвигов больше или равен размеру (в битах) операнда-приемника, для **SHL** и **SHR** флаг **CF** не определен.

OF

Флаг **OF** имеет смысл только для сдвигов на 1 бит. В противном случае он не определен.

- для сдвигов влево флаг **OF** устанавливается в 0, если старший значащий бит результата совпадает с флагом **CF** (то есть, старшие два бита исходного операнда были одинаковыми), в противном случае он равен 1.

- для инструкции **SAR** флаг **OF** сбрасывается для всех сдвигов на 1 бит.

- для инструкции **SHR** флаг **OF** устанавливается на самый значащий бит исходного операнда.

Флаги **SF**, **ZF** и **PF** устанавливаются в соответствии с результатом.

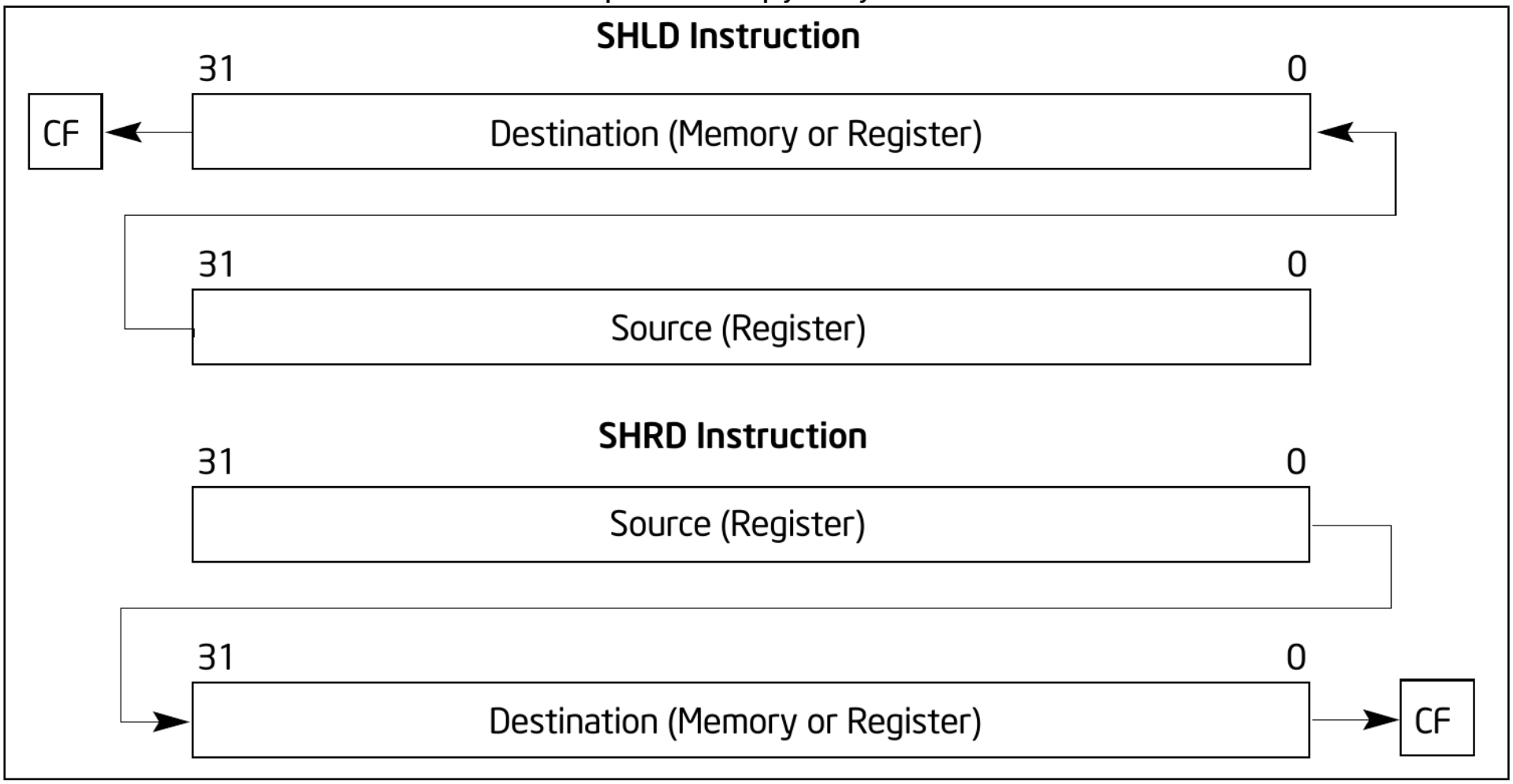
Если счетчик сдвига равен 0, флаги остаются без изменений.

Для ненулевого счетчика флаг **AF** не определен.

SHLD/SHRD — двойной сдвиг влево/вправо

Описание

Инструкции **SHLD** (сдвиг влево, дважды) и **SHRD** (сдвиг вправо, дважды) предназначены для выполнения многократных сдвигов влево и вправо, соответственно, на 64 бит и более — они сдвигают указанное число битов от одного операнда к другому.



Синтаксис

```
SHxD    r/m, r, imm8  ; 16, 32, x — L/R
SHxD    r/m, r, CL    ; 16, 32, x — L/R
```

Данные инструкции предоставляются для облегчения операций с невыровненными битовыми строками, а также могут быть использованы для реализации различных операций перемещения битовых строк.

Инструкция SHLD сдвигает влево первый операнд (операнд-приемник) на количество бит, указанное третьим операндом (операнд-счетчик). Второй операнд (операнд-источник), сдвигается влево, предоставляет выталкиваемые биты в освобождающиеся справа позиции (начиная с бита 0) операнда-назначения.

Операндом-приемником может быть регистр или ячейка памяти.

Операнд-источник — только регистр.

Операнд-счетчик — целое число без знака, которое может быть байтом непосредственного значения или регистром **CL**.

Если операндом-счетчиком является **CL**, размер сдвига равен логическому И содержимого регистра **CL** и маски счетчика.

В не 64-битных режимах и в 64-битном режиме по умолчанию используются только 5 бит — от 0 до 4. Они маскируют длину сдвига до значения от 0 до 31. Если длина сдвига больше, чем размер операнда, результат операции не определен.

Последний бит, вытолкнутый из операнда-назначения, попадает в **CF**.

Флаги

Если счетчик равен 1 или больше, флаг **CF** заполняется последним битом, вытолкнутым из операнда-назначения, а флаги **SF**, **ZF**, **PF** останавливаются в соответствии со значением результата.

Если в процессе 1-битного сдвига произошло изменение знака, устанавливается флаг **OF**, в противном случае он очищается.

Если операнд-счетчик равен 0, флаги остаются без изменений.

Для сдвигов, превышающих 1 бит, флаг **OF** не определен.

Если сдвиг имеет место (операнд-счетчик не равен 0), флаг **AF** не определен.

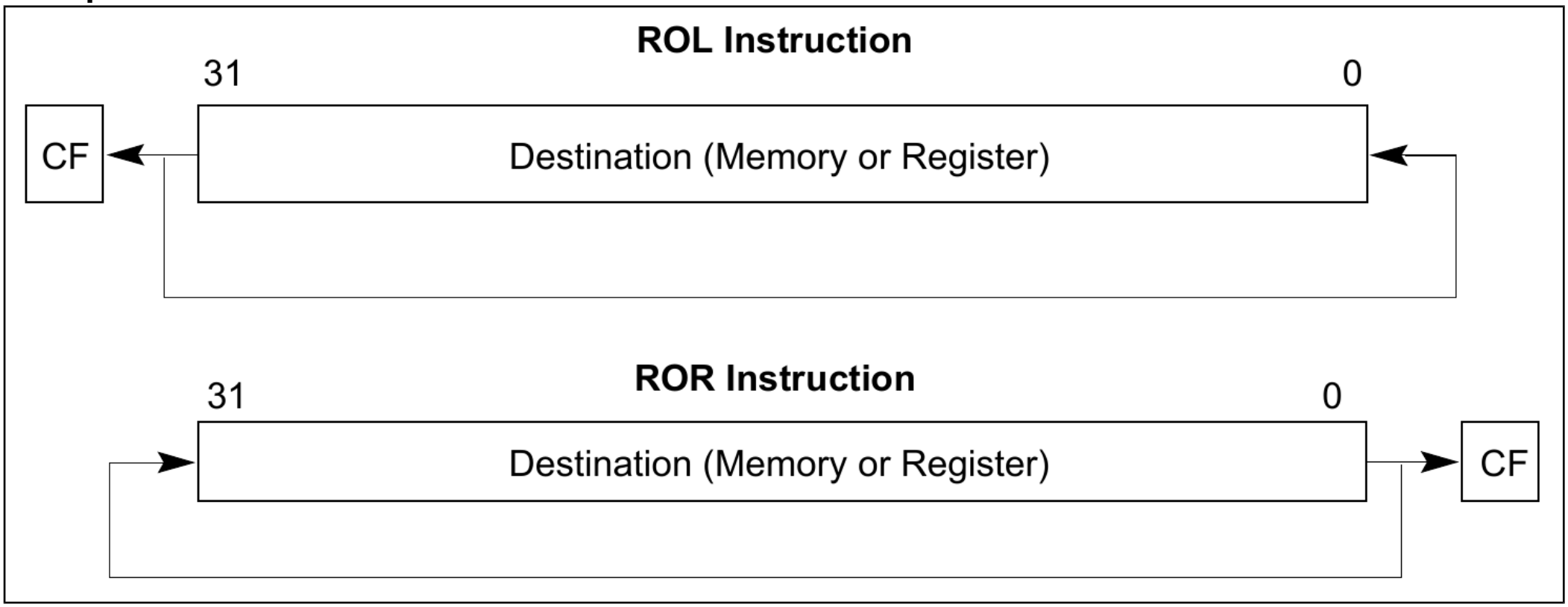
Если длина сдвига больше, чем размер операнда, флаги не определены.

ROL/ROR/RCL/RCR — циклические сдвиги

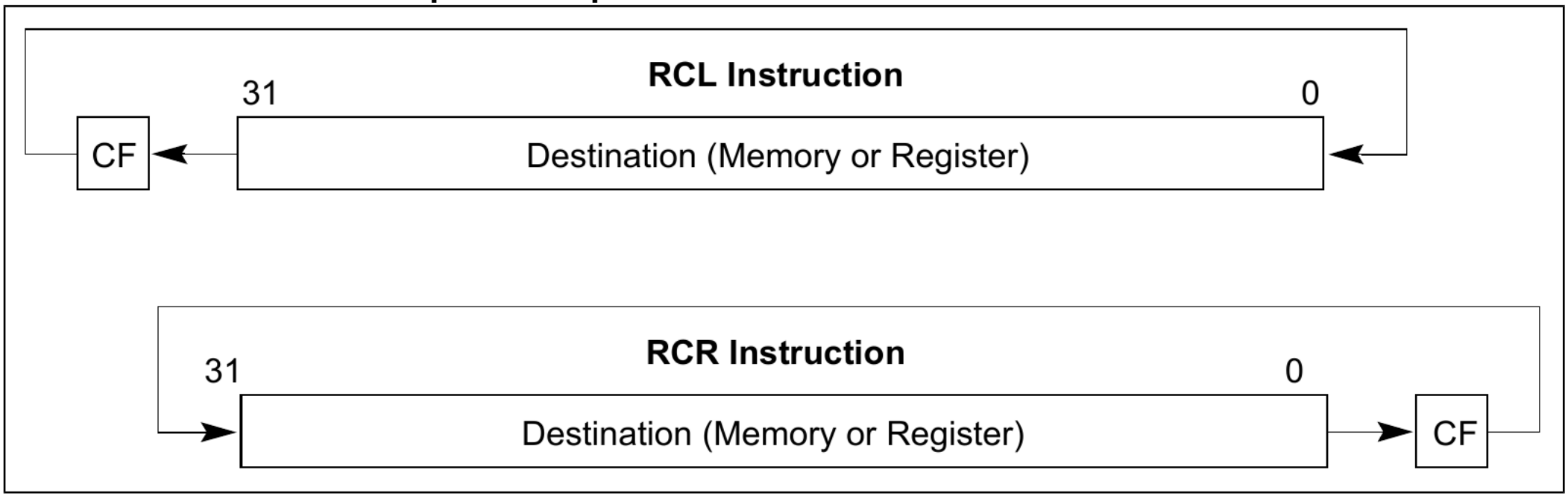
Описание

Инструкции циклически сдвигают (вращают) биты первого операнда (операнд-приемник) на количество битовых позиций, указанных во втором операнде (операнд-счетчик), и сохраняет результат в операнде-приемнике.

Простые циклические сдвиги



Циклические сдвиги через бит переноса



Инструкции циклического сдвига влево (**ROL**) и циклического сдвига влево через бит переноса **CF** (**RCL**) сдвигают все биты в сторону более старших разрядов, за исключением старшего разряда, который переносится в положение младшего разряда.

Команды циклического сдвига вправо (**ROR**) и циклического сдвига вправо через бит переноса **CF** (**RCR**) сдвигают все биты в сторону младших разрядов, за исключением младшего разряда, который переносится в положение старшего разряда.

Синтаксис

```
Rxy    r/m, 1    ; 8, 16, 32
Rxy    r/m, CL   ; 8, 16, 32
Rxy    r/m, imm8 ; 8, 16, 32
```

x — C (циклический через бит переноса), O (циклический)

y — L (влево), R (вправо)

Операндом-приемником может быть регистр или ячейка памяти.

Операнд-счетчик — целое число без знака, которое может быть непосредственным значением, или значением в регистре CL.

В традиционном режиме и режиме совместимости процессор ограничивает счетчик числом от 0 до 31, маскируя все биты в операнде счетчика, кроме 5 младших разрядов.

Инструкции **RCL** и **RCR** вовлекают в циклический сдвиг флаг **CF**.

Инструкция **RCL** сдвигает флаг **CF** в младший бит и сдвигает самый старший бит в флаг **CF**.

Инструкция **RCR** сдвигает флаг **CF** в старший бит и сдвигает младший бит в флаг **CF**. Для инструкций **ROL** и **ROR** исходное значение флага **CF** не является составляющей частью результата, но флаг **CF** в любом случае получает копию бита, который был перемещен с одного конца в другой.

Флаг **OF** имеет смысл только для 1-битных сдвигов и он не определен во всех других случаях (кроме только инструкций **RCL** и **RCR** — сдвиг на 0 бит ничего не делает, соответственно, не влияет и на флаги).

Для сдвигов влево флаг **OF** устанавливается в значение, равное операции исключающего ИЛИ для бита **CF** (после сдвига) и самого старшего бита результата.

Для правого «вращения» флаг **OF** устанавливается в значение, равное операции исключающего **ИЛИ** для двух старших значащих битов результата.

Флаг **CF** содержит значение сдвинутого в него бита.

Флаг **OF** затрагивается только для однобитовых сдвигов. Он не определен для многобитовых вращений.

Флаги **SF**, **ZF**, **AF**, **PF** остаются без изменений.

Инструкции для работы с битами и байтами

Битовые инструкции позволяют проверить и изменить отдельные биты в операндах типа слова (полуслова) и двойного слова (слова).

Байтовые инструкции позволяют задать значение байтовому операнду, чтобы отразить состояние флагов в регистре **EFLAGS**.

BT — проверить состояние бита (копировать бит в CF);
BTS/BTR/BTC — копировать бит в CF и установить/сбросить/инвертировать;
BSF/BSR — сканировать в поисках установленного бита;
SETcc — условная установка байта;
TEST — логическое сравнение;
POPCNT — количество установленных бит.

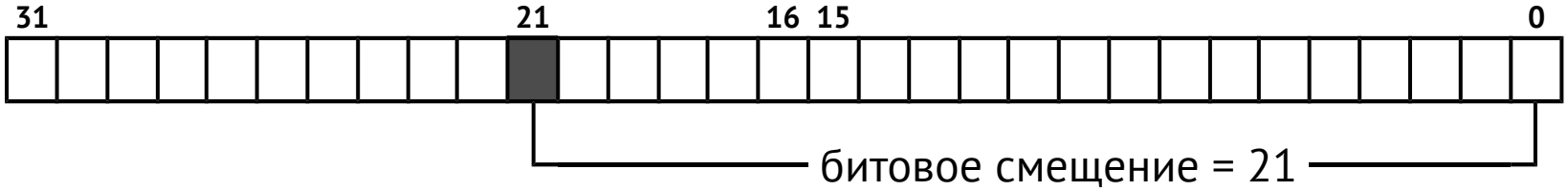
BT — проверить состояние бита

```
BT    r/m, r    ; 16, 32
BT    r/m, imm8 ; 16, 32
```

Операция

```
CF ← Bit(BitBase, BitOffset)
```

Выбирает бит в битовой строке, указанной с помощью первого операнда, называемого битовой базой (BitBase), в позиции, указанной вторым операндом, называемым битовым смещением (BitOffset), и сохраняет значение бита в флаге **CF**.



Операнд битовой базы может быть регистром или ячейкой памяти. Операнд битового смещения может быть регистром или непосредственным значением:

- если операнд битовой базы задан регистром, команда использует значение операнда битового смещения по модулю 16 или 32 (модуль зависит от режима и размера регистра).
- если операнд битовой базы указан как местоположение в памяти, он представляет адрес байта в памяти, содержащего битовую базу (бит 0 указанного байта) строки битов. Диапазон положений бита, на который может ссылаться операнд битового смещения, зависит от размера операнда.

Некоторые ассемблеры поддерживают битовые смещения большие, чем 31, если их указать в поле непосредственного операнда в сочетании с операндом битовой базы в памяти. В этом случае младшие 3 или 5 битов (3 для 16-битных операндов, 5 для 32-битных операндов) непосредственного битового смещения сохраняются в поле непосредственного значения, а старшие биты сдвигаются и комбинируются со смещением байта битовой базы, если это допускает режиме адресации. **Процессор же будет игнорировать старшие биты, если они не равны нулю.**

При обращении к биту в памяти в ряде случаев процессор может запросить доступ к четырем или двум байтам. Это имеет место, даже если для получения заданного бита требуется доступ только к одному байту.

Поэтому программное обеспечение должно избегать ссылок на области памяти, близкие к дырам в адресном пространстве. В частности, следует избегать ссылок на регистры ввода-вывода с отображением в памяти. Вместо этого программное обеспечение должно использовать инструкции **MOV** для загрузки или сохранения по этим адресам и использовать регистровую форму этих инструкций для манипулирования данными.

Флаг **CF** содержит значение выбранного бита.

Флаг **ZF** не изменяется.

Состояние флагов **OF**, **SF**, **AF**, **PF** не определено.

BTS/BTR/BTC/ — проверить бит и его установить/сбросить/инвертировать

```
Btx    r/m, r    ; 16, 32
Btx    r/m, imm8 ; 16, 32
```

Операция

```
CF ← Bit(BitBase, BitOffset)
Bit(BitBase, BitOffset) ← (Op)Bit(BitBase, BitOffset)
```

Выбирает бит в битовой строке, указанной с помощью первого операнда (BitBase), в позиции, указанной вторым операндом (BitOffset), сохраняет значение бита в флаге **CF**, после чего устанавливает его в 1 (**BTS**), сбрасывает в 0 (**BTR**) или инвертирует (**BTC**).

Операнд битовой базы может быть регистром или ячейкой памяти. Операнд битового смещения может быть регистром или непосредственным значением:

- если операнд битовой базы задан регистром, команда использует значение битового смещения по модулю 16 или 32 в зависимости от режима и размера регистра.

- если операнд битовой базы указан как местоположение в памяти, он представляет адрес байта в памяти, содержащего битовую базу (бит 0 указанного байта) строки битов. Диапазон положений бита, на который может ссылаться операнд битового смещения, зависит от размера операнда. Данная инструкция может использоваться с префиксом **LOCK**, чтобы обеспечить атомарность ее выполнения.

Флаг **CF** содержит измененное значение выбранного бита.

Флаг **ZF** не изменяется.

Состояние флагов **OF**, **SF**, **AF**, **PF** не определено.

BSF/BSR — сканировать в поисках установленного бита

```
Bsx    r, r/m    ; 16, 32
```

Описание

Сканирует операнд-источник (второй операнд) в поисках наименее значимого установленного бита (**BSF**) или наиболее значимого (**BSR**).

Если такой бит будет найден, его битовый индекс сохраняется в операнде-приемнике (первый операнд).

Операндом-источником может быть регистр или содержимое некоторого адреса в памяти. Операнд-приемник — регистр.

Битовый индекс — беззнаковое смещение относительно бита 0 операнда-источника. Если содержимое операнда-источника равно нулю, содержимое операнда-приемника неопределено.

Флаги

Флаг **ZF** устанавливается в единицу, если операнд-источник равен нулю, в противном случае флаг **ZF** сбрасывается в 0.

Состояние флагов **CF**, **OF**, **SF**, **AF**, **PF** не определено.

SETcc — установить байт, если условие выполнено

```
SETcc     r/m8
```

Описание

Устанавливает операнд-приемник в 0 или 1 в зависимости от установленных флагов состояния (**CF**, **SF**, **0F**, **ZF**, **PF**) в регистре **EFLAGS**.

Операнд-приемник указывает либо на регистр, либо на местоположение в памяти.

Суффикс условного кода (**cc**) в мнемонике инструкции указывает, какое условие проверяется.

cc	**Условие**		
E / Z	**E**qual / **Z**ero	равно / ноль	ZF=1
NE / NZ	**N**ot **E**qual / **N**ot **Z**ero	не равно / не ноль	ZF=0
A / NBE	**A**bove / **N**ot (**B**elow or **E**qual)	выше / не (ниже или равно)	CF=0 & ZF=0
AE / NB	**A**bove or **E**qual / **N**ot **B**elow	выше или равно / не ниже	CF=0
B / NAE	**B**elow / **N**ot (**A**bove or **E**qual)	ниже/ не (выше или равно)	CF=1
BE / NA	**B**elow or **E**qual / **N**ot **A**bove	ниже или равно / не выше	CF=1 | ZF=1
G / NLE	**G**reater / **N**ot (**L**ess or **E**qual)	больше / не (меньше или равно)	ZF=0 & SF=OF
GE / NL	**G**reater or **E**qual / **N**ot **L**ess	больше или равно / не меньше	SF=OF
L / NGE	**L**ess / **N**ot **G**reater or **E**qual	меньше / не (больше или равно)	SF≠OF
LE / NG	**L**ess or **E**qual / **N**ot **G**reater	меньше или равно / не больше	ZF=1 | SF≠OF

cc	**Условие**		
C	**C**arry	перенос	CF=1
NC	**N**o **C**arry	отсутствие переноса	CF=0
O	**O**verflow	переполнение	OF=1
NO	**N**o **O**verflow	отсутствие переполнения	OF=0
S	**S**ign (negative)	знак	SF=1
NS	**N**o **S**ign (non-negative)	отсутствие знака	SF=0
P/PE	**P**arity / **P**arity **E**ven	паритет / четно	PF=1
NP/PO	**N**o **P**arity / **P**arity **O**dd	отсутствие паритета / нечетно	PF=0

Термины «выше» и «ниже» связаны с флагом **`CF`** и ссылаются на отношения между двумя без-знаковыми числами.

Термины «больше» и «меньше» связаны с флагами **`CF`** и **`OF`** и ссылаются на отношения между двумя числами со знаком.

Некоторые инструкции **`SETcc`** имеют альтернативную мнемонику. Например, **`SETG`** (установить байт, если больше) и **`SETNLE`** (установить байт, если не (меньше или равно)) имеют один и тот же код операции и проверяют одно и то же условие — **`ZF == 0 && SF == OF`**. Данные мнемоники приводятся с целью сделать код более понятным.

Некоторые языки представляют логическую единицу (TRUE) как целое, у которого все биты установлены в 1.

Данное представление может быть получено выбором логически противоположного условия в инструкции **SETcc** и декремента результата.

Например, для проверки на переполнение (cc = **0**) вместо **SET0** следует использовать инструкцию **SETNO**, после чего декрементировать результат.

```
1 - 1 -> 0              FALSE
0 - 1 -> -1 (FFFFFFFF)  TRUE
```

Инструкция **SET0** дает для переполнения и его отсутствия значения 1 и 0.

Инструкция **SETNO** дает в этом случае 0 и 1, которые после декременитирования превращаются в -1 и 0, соответственно, что удовлетворяет требованиям представления логической единицы как целого, у которого все биты установлены в 1.

Флаги

Состояние флагов не изменяется.

TEST — Логическое сравнение

```
TEST    AL, imm8
TEST    AX, imm16
TEST    EAX, imm32

TEST    r/m, imm  ; 8, 16, 32
TEST    r/m, r    ; 8, 16, 32
```

Описание

Вычисляет побитовое логическое **И** (**AND**) первого операнда (операнд-источник 1) и второго операнда (операнд-источник 2) и устанавливает флаги состояния **SF**, **ZF** и **PF** в соответствии с результатом. Результат затем отбрасывается.

Флаги

Флаги **OF** и **CF** сбрасываются в 0.

Флаги **ZF**, **SF**, **PF** устанавливаются в соответствии с результатом.

Состояние флага **AF** неопределено.

POPCNT — количество установленных бит

```
POPCNT    r, r/m    ; 16, 32
```

Описание

Инструкция вычисляет количество бит, установленных во втором операнде (операнд-источник) в состояние 1, и возвращает его в первом (регистр назначения).

Флаги

Флаги **OF**, **SF**, **ZF**, **AF**, **CF**, **PF** очищаются.

Флаг **ZF** устанавливается, если SRC = 0, в противном случае **ZF** очищается.