Пересылки, Команды MOV/XCHG, оператор PTR

Мы переходим к изучению команд ПК и способов их записи в ЯА. При описании команд нам придётся указывать, какие операнды в них допустимы, а какие – нет. Для сокращения подобного рода указаний вводятся специальные обозначения, которыми пользуются при описании команд.

Местонахождение операнда	*Обозначение*	*Запись в ЯА*
В команде	i8, i16	Константное выражение
В регистре общего назначения	r8, r16	Имя регистра
В сегментном регистре (кроме CS)	sr	ds,ss,es
В ячейке памяти	m8, m16	Адресное выражение

Непосредственный операнд (задаваемый в самой команде) обозначается буквой i (immediate, непосредственный), указывая за ней, сколько разрядов – 8 (байт), 16 (слово) или 32 (двойное слово) отводится на него в команде. При этом отметим, что тут указывается максимальное число разрядов. То есть, сам операнд может быть меньше либо равен заданному числу.

Регистры обозначаются буквой r (от register), указывая за ней требуемый размер регистра: r8 – это байтовые регистры (ah,al,bh и тд). При этом, данные регистры относятся к регистрам общего назначения. Сегментные регистры обозначаются sr.

MOV

В ПК существую команды пересылки байта или слова (переслать одной командой двойное слово нельзя (8086). Пересылаемая величина берётся из регистра или ячейки памяти и записывается в ячейку памяти или регистр.

mov op1, op2 (op1 = op2)

На флаги данная команда не влияет. В команде MOV допустимы следующие операции:

op1	*op2*	
r8	i8, r8, m8	пересылка байтов
m8	i8, r8	
r16	i16, r16, sr, m16	пересылка слов
sr (кроме CS)	r16, m16	
m16	i16, r16, sr	

Тут видно, что запрещены пересылки из одной ячейки памяти в другую, из одного сегментного регистра в другой и запись непосредственного операнда в сегментный регистр. Это обусловлено тем, что в ПК нет таких машинных команд. Если алгоритму всё же необходимо такое действие, что можно выполнить его за две команды. Регистр cs нельзя менять потому, что его содержимое обозначает адрес сегмента программы и, его изменение, фактически считается переходом. Команды перехода мы будем рассматривать позже.

Как известно, в ПК числа размером в слово хранятся в памяти в «перевёрнутом» виде, а в регистрах – в нормальном. Команда mov учитывает это и при пересылке слов между памятью и регистрами сама «переворачивает» их.

```
Q DW 1234h
        Mov ax, Q          ;ah = 12h, al = 34h
```

По команде mov можно переслать байт или слово. При пересылке, размеры обоих операндов должны совпадать, иначе ассемблер зафиксирует ошибку. Когда в операции пересылки участвует указатель на память, то размер этого операнда определяется исходя из типа данного указателя. Но очень часто указатели хранятся в регистрах, а, следовательно, ассемблеру определить тип не получится. В этом случае, программист должен сам, явно указать его. Делается это с помощью оператора указания типа **PTR**. Например,

```
mov [si], 0
```

Тут нельзя понять, какого размера 0 пересылается, байта или слова. Тоже касается и первого операнда. Непонятно, записать нужно 0 в байт по адресу si или в слово. В подобных ситуациях ассемблер фиксирует ошибку. Чтобы избежать этой ошибки, существует вышеупомянутый оператор.
<тип> PTR <выражение>
Если оператор указания типа относится ко второму операнду
Mov [si], byte ptr 0
То он [оператор] будет константным выражением. В данном случае считается, что 0 представляется как байт. Кстати, конструкции вида byte ptr 300 или word ptr 70000 будут вызывать ошибки, так как число 300 не может влезть в один байт, так же, как и число 70000 не влезает в слово. Если оператор относится к первому операнду, то оператор относится к адресным выражениям. Обычно принято уточнять тип операнда-адрес.

xchg

В машинных командах довольно часто приходится переставлять операнды местами. Конечно, такую перестановку можно выполнить и с помощью mov, но несмотря на это, в ПК введена специальная команда:
xchg op1,op2
Допустимые типы операндов:

op1	*op2*	
r8	r8, m8	перестановка байтов
m8	r8	
r16	r16, m16	перестановка слов
m16	r16	

Как видно, что не допускается переставлять местами две ячейки памяти. Если очень нужно, то можно это сделать за несколько команд.

Арифметические операции

Беззнаковые числа складываются как обычно, только в двоичной системе счисления. Однако, бывают ситуации, когда сумма получается большой и не влезает в отведённую ячейку. Например, при сложении 250 и 10 получим 260 (**1**00000100), которое не влезет в байт. В некоторых ЭВМ в такой ситуации фиксируется ошибка и на этом прекращается выполнение программы. Однако, в 8086 предусмотрена другая ситуация: ошибка не фиксируется, левая единица (единица переноса) отбрасывается и в качестве ответа выдаётся то, что влезло – 00000100b или 4. Но, зато, невлезщая единица переходит в флаг CF. Если при сумме флаг CF установился в 1, то это значит, что произошёл перенос. Кстати, суммирование с отбрасыванием единицы переноса – суммирование по модулю k, где k – размер ячейки.

$$\text{сумма}(x,y)=(x+y) \bmod 2^k=\begin{cases} x+y, & \text{если } x+y< 2^k,\ CF=0 \\ x+y-2^k, & \text{если } x+y>=2^k,\ CF=1 \end{cases}$$

При вычитании беззнаковых чисел ситуация похожа. Что делать, если при вычитании x и y, у нас x < y? По идее, результат получится отрицательный, но он вне зоны действия беззнаковых чисел. В таком случае делается вот что, если x и правду < y, тогда числу x даётся заём единицы, то есть к x прибавляется 2^k, и только после этого производится вычитание. Полученная разность и будет результатом. Например, при k=8, x=1, y=2 разность будет такая:
1-2 (1+2^8)-2 = 257-2 = 255
Ошибка тут не фиксируется, зато флаг CF устанавливается. Вычитание такое называется вычитанием по модулю 2^k.

$$\text{разность}(x,y)=(x-y) \bmod 2^k=\begin{cases} x-y, & \text{если } x>=y, \quad CF=0 \\ (2^k+x)-y, & \text{если } x<y, \quad CF=1 \end{cases}$$

То есть, при сложении и вычитании беззнаковых чисел нужно учитывать флаг CF. Если был заём или перенос единицы, то он будет установлен. В противном случае, флаг CF – сбрасывается. Сами операции выполняются по модулю 2^k.

При сложении и вычитании знаковых чисел делается следующее: числа представляются как будто они беззнаковые, производится соответствующая операция и, от неё берётся модуль по 2^k. Пример: 3 + (-1). Их дополнительные коды: 3 и 2^k-1 = 255, при k = 8. Получается (3+255) % 256 = 258 mod 255 = 2. Или -3 + 1 (256-3) + 1= 253 + 1 = 254 = -2 (254=256-2).
При сложении и вычитании знаковых чисел нужно учитывать переполнение мантиссы. Например, 127 + 2 = 129. Рассмотрим его как дополнительный код 129 = (256-127). То есть, если сложить 127 и 2 то получим -127. Ошибка не будет зафиксирована, но, флаг OF установится в единицу.
Дополнительный код отрицательного числа – инвертировать биты модуля числа + 1. Пример при вычитании:
127 – (-2) = 127+2 = 129. А это дополнительный код 256-**127.**
Сложение и вычитание знаковых или беззнаковых чисел комп выполняет одними и теми же командами. Поэтому, одновременно фиксируются оба флага (CF/OF). Программист сам должен знать на какие именно ему обращать внимание.
ADD op1, op2
SUB op1, op2

op1	*op2*	
r8	i8, r8, m8	сложение/вычитание байтов
m8	i8, r8	
r16	i16, r16, m16	сложение/вычитание слов
m16	i16, r16	

В этих командах меняются регистры CF,OF,SF,ZF,AF и PF.
add op1, op2 - сложение
Команда выполняет арифметическое сложение приемника и источника, помещает сумму в приемник, не изменяя содержимое источника. Приемник может быть регистром или переменной, источник - числом, регистром или переменной, но нельзя использовать переменную одновременно и для источника, и для приемника. Команда ADD никак не различает числа со знаком и без знака, но, употребляя значения флагов CF (перенос при сложении чисел без знака), OF (перенос при сложении чисел со знаком) и SF (знак результата), разрешается применять ее и для тех, и для других.
sub op1, op2 - вычитание
Вычитает источник из приемника и помещает разность в приемник. Приемник может быть регистром или переменной, источник - числом, регистром или переменной, но нельзя использовать переменную одновременно и для источника, и для приемника. Точно так же, как и команда ADD, SUB не делает различий между числами со знаком и без знака, но флаги позволяют использовать её и для тех, и для других.

Изменение на 1:
INC op
DEC op

r8, m8, r16, m16.

inc op1 - инкремент
Увеличивает приемник (регистр или переменная) на 1. Единственное отличие этой команды от ADD приемник,1 состоит в том, что флаг CF не затрагивается. Остальные арифметические флаги (OF, SF, ZF, AF, PF) устанавливаются в соответствии с результатом сложения.
dec op1 - декремент
Уменьшает приемник (регистр или переменная) на 1. Единственное отличие этой команды от SUB приемник,1 заключается в том, что флаг CF не затрагивается. Остальные арифметические флаги (OF, SF, ZF, AF, PF) устанавливаются в соответствии с результатом вычитания.
Изменение знака:
NEG OP

r8, m8, r16, m16.

Команда рассматривает свой операнд как число со знаком и изменяет его знак.
Выполняет над числом, содержащимся в приемнике (регистр или переменная), операцию дополнения до двух. Эта операция эквивалентна обращению знака операнда, если рассматривать его как число со знаком. Если приемник равен нулю, флаг CF устанавливается в 0, иначе - в 1. Остальные флаги (OF, SF, ZF, AF, PF) назначаются в соответствии с результатом операции.

```
Mov ah, 1
NEG AH        ;AH = -1 (0xFF)
```

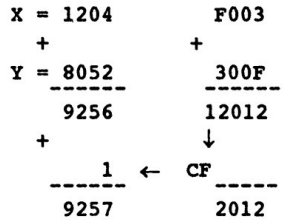
Есть особый случай: если op = -128 и находится в байтовом типе данных, то операнд не меняется, так как в байте нет числа 128. Аналогично и с другими типами. В этом случае, установится флаг OF (в других случаях OF сбрасывается).
Красивый пример использования команды NEG - получение абсолютного значения числа, применяя всего две команды - изменение знака и переход на первую команду еще раз, если знак отрицательный:

```
label0: neg eax
        js label0
```

Сложение/вычитание с учётом переноса/заёма. Допустимые типы операндов как и при ADD/SUB. ADC аналогична ADD, только к сумме операндов ещё прибавляется значение флага CF. OP1 = op1+op2+CF. А в команде разности SBB вычитается ещё 1. Op1 = op1-op2 – CF.
adc op1, op2 - сложение с переносом
Эта команда аналогична ADD, но при этом выполняет арифметическое сложение приемника, источника и флага CF. Пара команд ADD/ADC используется для сложения чисел повышенной точности. Сложим, например, два 32-битных целых числа. Пусть одно из них находится в паре регистров DX:AX (младшее слово (биты 0-15) - в AX и старшее (биты 16-31) - в DX), а другое - в паре регистров BX:CX:

```
add ax, cx
adc dx, bx
```

Если при сложении младших двойных слов произошел перенос из старшего разряда (флаг CF = 1), то он будет учтен следующей командой ADC.



sbb op1, op2 - вычитание с займом
Эта команда аналогична SUB, но она вычитает из приемника значение источника и дополнительно вычитает значение флага CF. Ее можно использовать для вычитания 32-битных чисел в DX:AX и BX:CX аналогично ADD/ADC:
sub ax, cx
sbb dx, bx
Если при вычитании младших двойных слов произошел заем, то он будет учтен при вычитании старших слов.
Команды умножения и деления.
Умножение:

```
умножение целых без знака (multiply):          MUL op
умножение целых со знаком (integer multiply): IMUL op

умножение байтов: AX:=AL*op              (op: r8, m8)
умножение слов:   (DX,AX):=AX*op         (op: r16, m16)
```

MUL: Выполняет умножение содержимого источника (регистр или переменная) и регистра AL, AX (в зависимости от размера источника) и помещает результат в AX, DX:AX соответственно. Если старшая половина результата (AH, DX) содержит только нули (результат целиком поместился в младшую половину), флаги CF и OF устанавливаются в 0, иначе - в 1. Значение остальных флагов (SF, ZF, AF и PF) не определено.
IMUL: источник (регистр или переменная) умножается на AL или AX (в зависимости от размера операнда), и результат располагается в AX или DX:AX.
Во всех вариантах считается, что результат может занимать в два раза больше места, чем размер источника, поскольку существует вероятность переполнения и потери старших битов результата. Флаги OF и CF будут равны единице, если это произошло, и нулю, если результат умножения поместился целиком в младшую половину приемника.
Значения флагов SF, ZF, AF и PF после команды IMUL не определены.
Как видно из описания, в качестве операнда не может выступать непосредственный операнд. Первый сомножитель не указывается, так как он должен находиться всегда в регистре ax/al. Результат умножение записывается в регистр ax, если операнд – байт и в dx:ax, если операнд – слово. Старшая часть находится в AH/DX, а младшая в AL/AX.

```
N DB 10
  ...
  MOV AL,2
  MUL N          ; AX=2*10=20=0014h: AH=00h, AL=14h
  MOV AL,26
  MUL N          ; AX=26*10=260=0104h: AH=01h, AL=04h
  MOV AX,8
  MOV BX,-1
  IMUL BX        ; (DX,AX)=-8=0FFFFFFF8h: DX=0FFFFh, AX=0FFF8h
```

То есть, результат хранится в удвоенном формате. Это не всегда пригождается. Иногда известно, когда результат может быть такого-же размера как его операнд, а иногда нет. Возникает вопрос, когда нужно учитывать старшую часть? Ответ в том, что если результат произведения больше операнда, то CF=OF устанавливаются оба. Иначе сбрасываются.

Деление – как и умножение реализуется в две команды, в зависимости от из «знаковости».

Деление целых без знака (divide): **DIV op**
Деление целых со знаком (integer divide): **IDIV op**

деление слова на байт:

AH:=AX mod op, AL:=AX div op **(op: r8, m8)**

деление двойного слова на слово:

DX:=(DX,AX) mod op, AX:=(DX,AX) div op (op: r16, m16)

DIV: Выполняет целочисленное деление без знака AX или DX:AX (в зависимости от размера источника) на источник (регистр или переменная) и помещает результат в AL или AX, а остаток - в AH или DX соответственно. Результат всегда округляется в сторону нуля, абсолютное значение остатка меньше абсолютного значения делителя. Флаги CF, OF, SF, ZF, AF и PF после этой команды не определены, а переполнение или деление на ноль вызывает исключение #DE (ошибка при делении) в защищенном режиме и прерывание О-в реальном.
IDIV: Выполняет целочисленное деление со знаком AX или DX:AX (в зависимости от размера источника) на источник (регистр или переменная) и помещает результат в AL или AX, а остаток - в AH или DX соответственно. Результат всегда округляется в сторону нуля, знак остатка совпадает со знаком делимого, абсолютное значение остатка меньше абсолютного значения делителя.
Флаги CF, OF, SF, ZF, AF и PF после этой команды не определены, а переполнение или деление на ноль вызывает исключение #DE (ошибка при делении) в защищенном режиме и прерывание О - в реальном.
Местонахождение первого операнда (делимого) и результата фиксировано и потому явно не указывается. Указывается только второй операнд (делитель), который может находиться в регистре или в ячейке памяти, но не может быть непосредственным операндом. При делении слова на байт, делимое находится в AX, а делитель должен быть байтом. При делении двойного слова на слово – делитель должен быть в DX:AX. Старшая часть делителя в DX, а младшая в AX. В результате, на место делимого запишется остаток в старшую часть, а в младшую – целая часть. Есть случаи, когда деление невозможно. Это при делении на 0, или, когда результат не помешается в ячейку. Например:

```
MOV AX,600
MOV BH,2
DIV BH  ;
```

300 не влезет в AL. В таких ситуациях ПК прекращает выполнение программы.

Расширение слова до двойного слова.
При беззнаковых числах, достаточно нанулить старшую часть – регистр DX. При знаковых:
cwd - конвертирование слова в двойное слово
Команда CWD превращает слово в AX в двойное слово, младшая половина которого (биты 0-15) остается в AX, а старшая (биты 16-31) располагается в DX.
Эта команда лишь устанавливает все биты регистра DX в значение, равное величине старшего бита регистра AX, сохраняя таким образом его знак.
Флаги не меняются.
Расширение байта до слова.
Для беззнаковых чисел достаточно занулить старшую часть. Для чисел со знаком:
cbw - конвертирование байта в слово
CBW расширяет байт, находящийся в регистре AL, до слова в AX.

Так же как и в команде CWD, расширение выполняется путем установки каждого бита старшей половины результата равным старшему биту исходного байта или слова, то есть:
mov al,OF5h ; AL = OF5h = 245 = -11.
Cbw ; Теперь AX = OFFF5h = 65 525 = -11.
Флаги не меняются.

Логические операции
AND, OR, NOT, XOR
and op1, op2 - логическое И
Команда выполняет побитовое «логическое И» над приемником (регистр или переменная) и источником (число, регистр или переменная; источник и приемник не могут быть переменными одновременно) и помещает результат в приемник. Любой бит результата равен 1, только если соответствующие биты обоих операндов были равны 1, и равен 0 в остальных случаях. Наиболее часто AND применяют для выборочного обнуления отдельных битов. Например, команда
and al,00001111b
обнулит старшие четыре бита регистра AL, сохранив неизменными четыре младших.
Флаги OF и CF обнуляются, SF, ZF и PF устанавливаются в соответствии с результатом, AF не определен.
or op1, op2 - логическое ИЛИ
Выполняет побитовое «логическое ИЛИ» над приемником (регистр или переменная) и источником (число, регистр или переменная; источник и приемник не могут быть переменными одновременно) и помещает результат в приемник. Любой бит результата равен 0, только если соответствующие биты обоих операндов были равны 0, и равен 1 в остальных случаях. Команду OR чаще всего используют для выборочной установки отдельных битов. Например, команда
or al,00001111b
приведет к тому, что младшие четыре бита регистра AL будут установлены в 1.
При выполнении команды OR флаги OF и CF обнуляются, SF, ZF и PF устанавливаются в соответствии с результатом, AF не определен.
xor op1, op2 - логическое исключающее ИЛИ
Выполняет побитовое «логическое исключающее ИЛИ» над приемником (регистр или переменная) и источником (число, регистр или переменная; источник и приемник не могут быть переменными одновременно) и помещает результат в приемник. Любой бит результата равен 1, если соответствующие биты операндов различны, и нулю - в противном случае. XOR используется для самых разных операций, например:
xor ax,ax ; Обнуление регистра AX.
или
xor ax,bx
xor bx,ax
xor ax,bx ; Меняет местами содержимое AX и BX.
Оба примера могут выполняться быстрее, чем соответствующие очевидные команды
mov ax,0
или
xchg ax, bx
not op1 - инверсия
Каждый бит приемника (регистр или переменная), равный нулю, устанавливается в 1, и каждый бит, равный 1, сбрасывается в 0. Флаги не затрагиваются.
sar op1, op2 - арифметический сдвиг вправо
sal op1, op2 - арифметический сдвиг влево
shl op1, op2 - логический сдвиг вправо
shr op1, op2 - логический сдвиг влево

Эти четыре команды выполняют двоичный сдвиг приемника (регистр или переменная) вправо (в сторону младшего бита) или влево (в сторону старшего бита) на значение счетчика (число или регистр CL, из которого учитываются только младшие 5 бит, принимающие значения от 0 до 31). Операция сдвига на 1 эквивалентна умножению (сдвиг влево) или делению (сдвиг вправо) на 2. Так, число 0010b (2) после сдвига на 1 влево превращается в 0100b (4). Команды SAL и SHL выполняют одну и ту же операцию (на самом деле это одна и та же команда) - на каждый шаг сдвига старший бит заносится в CF, все биты сдвигаются влево на одну позицию, и младший бит обнуляется. Команда SHR осуществляет прямо противоположную операцию: младший бит заносится в CF, все биты сдвигаются на 1 вправо, старший бит обнуляется. Эта команда эквивалентна беззнаковому целочисленному делению на 2. Команда SAR действует по аналогии с SHR, только старший бит не обнуляется, а сохраняет предыдущее значение, вот почему, например, число 11111100b (-4) перейдет в 11111110b (-2). SAR, таким образом, эквивалентна знаковому делению на 2, но, в отличие от IDIV, округление происходит не в сторону нуля, а в сторону отрицательной бесконечности. Так, если разделить —9 на 4 с помощью IDIV, получится -2 (и остаток -1), а если выполнить арифметический сдвиг вправо числа -9 на 2, результатом будет -3. Сдвиги больше 1 эквивалентны соответствующим сдвигам на 1, выполненным последовательно.
Сдвиги на 1 изменяют значение флага OF: SAL/SHL устанавливают его в 1, если после сдвига старший бит изменился (то есть старшие два бита исходного числа не были одинаковыми), и в 0, если старший бит остался тем же. SAR устанавливает OF в 0, а SHR -- с значение старшего бита исходного числа. Для сдвигов на несколько битов значение OF не определено. Флаги SF, ZF, PF назначаются всеми сдвигами в соответствии с результатом, параметр AF не определен (кроме случая, когда счетчик сдвига равен нулю: ничего не происходит и флаги не изменяются).
В процессорах 8086 в качестве второго операнда можно было задавать лишь число 1 и при использовании CL учитывать все биты, а не только младшие 5, но уже начиная с 80186 эти команды приняли свой окончательный вид.

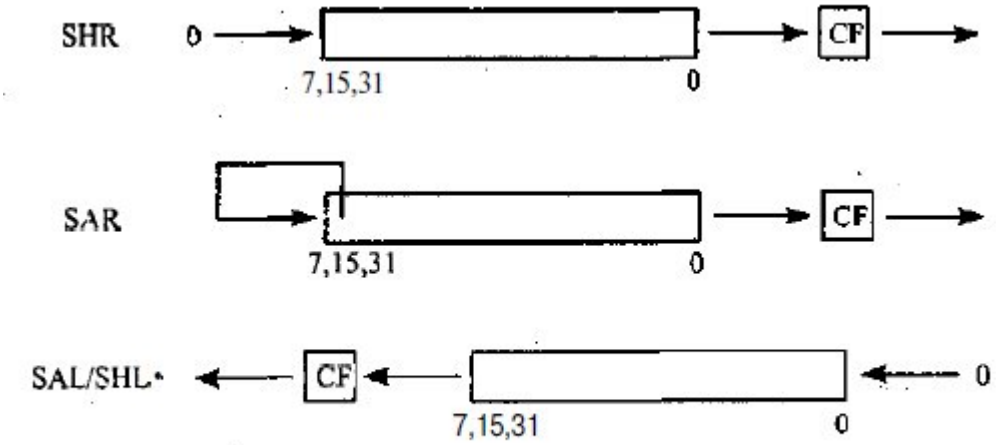


Рис. 7. Сдвиговые операции

ROR op1, op2 - циклический сдвиг вправо
ROL op1, op2 - циклический сдвиг влево
RCR op1, op2 - циклический сдвиг вправо через флаг переноса
RCL op1, op2 - циклический сдвиг влево через флаг переноса
Эти команды осуществляют циклический сдвиг приемника (регистр или переменная) на число битов, указанное в счетчике (число или регистр CL, из которого учитываются только младшие 5 бит, принимающие значения от 0 до 31). При выполнении циклического сдвига на 1 команды ROR (ROL) перемещают каждый бит приемника вправо (влево) на одну позицию, за исключением самого младшего (старшего), который записывается в позицию самого старшего (младшего) бита. Команды ,RCR и RCL выполняют аналогичное действие, но включают флаг CF в цикл, как если бы он был дополнительным битом в приемнике.

После выполнения команд циклического сдвига флаг CF всегда равен последнему вышедшему за пределы приемника биту, флаг OF определен только для сдвигов на 1 - он устанавливается, если изменилось значение самого старшего бита, и сбрасывается в противном случае. Флаги SF, ZF, AF и PF не изменяются.

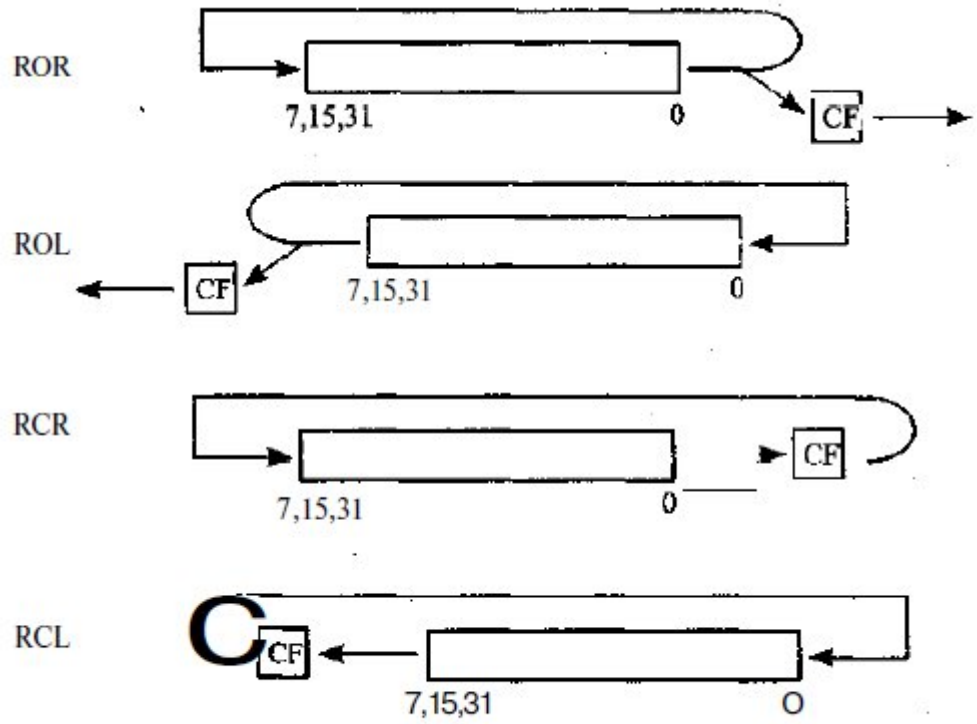


Рис. 9. Циклические сдвиги

Команды перехода

Прямой переход

JMP op - безусловный переход

JMP передает управление в другую точку программы, не сохраняя какой-либо информации для возврата. Операндом может быть непосредственный адрес для перехода (в программах используют имя метки, установленной перед командой, на которую выполняется переход), а также регистр или переменная, содержащая адрес.

В зависимости от типа перехода различают:

переход типа short (короткий переход) - если адрес перехода находится в пределах -128...+ 127 байт от команды JMP;

переход типа near (ближний переход) - если адрес перехода находится в том же сегменте памяти, что и команда JMP;

переход типа far (дальний переход) - если адрес перехода находится в другом сегменте. Дальний переход может выполняться и в тот же самый сегмент при условии, что в сегментной части операнда указано число, совпадающее с текущим значением CS;

переход с переключением задачи - передача управления другой задаче в многозадачной среде, в реальном режиме нас такой переход не интересует.

При выполнении переходов типа short и near команда JMP фактически преобразовывает значение регистра IP, изменяя тем самым смещение следующей исполняемой команды относительно начала сегмента кода. Если операнд - регистр или переменная в памяти, то его показатель просто копируется в IP, как если бы это была команда MOV. Если операнд для JMP - непосредственно указанное число, то его значение суммируется с содержимым EIP, приводя к относительному переходу. В ассемблерных программах в качестве операнда обычно указывают имена меток, но на уровне исполняемого кода ассемблер вычисляет и записывает именно относительные смещения.

Выполняя дальний переход в реальном, виртуальном и защищенном режимах (при переходе в сегмент с теми же привилегиями), команда JMP просто загружает новое значение в IP и новый селектор сегмента кода в CS, используя старшие 16 бит операнда как новое значение для CS и младшие 16 в качестве значения IP.
Op указывает на адрес перехода, т е на адрес команды которая будет выполнена следующей. При программировании на ЯА адрес нужной команды не надо самому высчитывать. Достаточно на неё поставить метку и ассемблер сам подставит нужный адрес. Адрес следующей команды хранится в IP регистре. Поэтому, чтобы совершить переход, его нужно изменить. Казалось бы, пусть бы команда jmp просто меняла бы это регистр и всё. Но делается немного иначе. В ПК в команде перехода указывается не полный адрес нужной команды, а разница между текущим адресом и адресом команды. (Если точно, то относительный адрес отсчитывается не от самой команды перехода, а от следующей за ней). В ПК существуют разные машинные команды перехода, в одной из которых относительный адрес перехода задаётся в виде байта (короткий переход), а в другой в виде слова (длинный переход). В каждой из них, операнд рассматривается как целое со знаком. Две команды существуют потому, что если бы мы указывали всегда длинный переход, то под операнд приходилось бы всегда отводить два байта. Однако, на практике, большинство переходов являются короткими, то есть переходы на команды, которые недалеко друг от друга. Если указывать разницу, то она будет помещаться в байт. В 8086 op не может быть непосредственным операндом. Короткий переход ассемблер подставляет по умолчанию только в том случае, если переход совершается назад. Только тогда он может высчитать разницу адресов и подставить полученной значение в качестве операнда. Но, если переход совершается вперёд, то ассемблер «подстраховывается» и вставляет полный адрес, так как заранее не может вычислить разницу. Если программист уверен, что переход вперёд короткий, то он может использовать оператор SHORT. Тогда ассемблер закодирует операнд размером в байт. Но, если мы ошиблись с SHORT, то ассемблер зафиксирует ошибку.

Косвенный переход

JMP r16 или JMP m16
Тут берётся содержимое регистра или адреса слова памяти и по нему совершается переход. Причём, этот адрес рассматривается как полный, а не отсчитанный от команды перехода.
Рассмотрим пример. Пусть у нас есть идентификатор L. Если он – метка, то будет совершён прямой переход. А если это имя переменной, то совершится косвенный переход по содержимому в этой ячейки памяти. Тут всё понятно, если L описано до команды перехода. Если после, то возникает неоднозначность. То есть ассемблер не будет знать, какой переход собирать, прямой или косвенный. Чтобы убрать неопределённость, ассемблер всегда считает, что L – метка. Если нам самим надо сделать косвенный переход, то нужно использовать уже известный нам оператор WORD PTR L.

Команды сравнения:
Cmp,test.
cmp op1, op2 - сравнение
Сравнивает приемник и источник и устанавливает флаги. Действие осуществляется путем вычитания источника (число, регистр или переменная) из приемника (регистр или переменная; приемник и источник не могут быть переменными одновременно), причем результат вычитания никуда не записывается.
Единственным следствием работы этой команды оказывается изменение флагов CF, OF, SF, ZF, AF и PF. Обычно команду CMP используют вместе с командами условного перехода (Jcc), а также условной пересылки данных (CMOVcc) или условной установки байтов (SETcc), правда две последние отсутствуют в наборе команд 8086. Эти команды позволяют применить результат сравнение, не обращая внимания на детальное значение каждого флага. Так, команда JE выполнит условный переход, если значения операндов предшествующей команды cmp были равны.
test op1, op2 - логическое сравнение
Вычисляет результат действия побитового «логического И» над приемником (регистр или переменная) и источником (число, регистр или переменная; источник и приемник не могут быть

переменными одновременно) и устанавливает флаги SF, ZF и PF в соответствии с полученным показателем, не сохраняя результата (флаги OF и CF обнуляются, значение AF не определено). TEST, так же как и CMP, используется в основном в сочетании с командами условного перехода (Jcc), условной пересылки данных (CMOVcc) и условной установки байтов (SETcc).
Несмотря на то что условные команды почти всегда вызываются сразу после CMP, не надо забывать, что их можно использовать после любой команды, модифицирующей флаги, например: проверить равенство AX нулю более короткой командой
test ax,ax
а равенство единице - однобайтной командой
dec ax
Условные переходы:
jcc op1 - условный переход
Это набор команд, выполняющих переход (типа short или near), если удовлетворяется соответствующее условие, которым в каждом случае реально является состояние тех или иных флагов. Но, когда команда из набора Jcc используется сразу после CMP, условия приобретают формулировки, соответствующие отношениям между операндами CMP. Например, если операнды CMP были равны, то команда JE, выполненная сразу после CMP, осуществит переход.
Операнд для всех команд из набора Jcc - 8-битное смещение относительно текущей команды.
Команды Jcc не поддерживают дальних переходов, поэтому, если требуется выполнить условный переход на дальнюю метку, необходимо использовать команду из набора Jcc с обратным условием и дальний JMP, как, например:
```
cmp ax,0
jne local_1
jmp far_label ; Переход, если AX = 0.
local_1:
```

Мнемокод	*Содержательное условие для перехода после CMP op1,op2*	*Состояние флагов для перехода*
	для любых чисел:	
JE	op1=op2	ZF=1
JNE	op1<>op2	ZF=0
	для чисел со знаком:	
JL/JNGE	op1<op2	SF<>OF
JLE/JNG	op1<=op2	SF<>OF или ZF=1
JG/JNLE	op1>op2	SF=OF и ZF=0
JGE/JNL	op1>=op2	SF=OF
	для чисел без знака:	
JB/JNAE	op1<op2	CF=1
JBE/JNA	op1<=op2	CF=1 или ZF=1
JA/JNBE	op1>op2	CF=0 и ZF=0
JAE/JNB	op1>=op2	CF=0

Мнемокод	*Условие перехода*	*Мнемокод*	*Условие перехода*
JZ	ZF=1	**JNZ**	ZF=0
JS	SF=1	**JNS**	SF=0
JC	CF=1	**JNC**	CF=0
JO	OF=1	**JNO**	OF=0
JP	PF=1	**JNP**	PF=0

Если нет n, то переход НУЖНО выполнить при установленном флаге. Если есть n, то переход нужно выполнить, если флаг СБРОШЕН.

Есть эквивалентные команды:
Jz/je, jne/jnz, jb/jc,jnb/jnc

Циклы
JCXZ op1 - переход, если CX=0
Выполняет ближний переход на указанную метку, если регистр CX равен нулю. Так же как и команды из серии Jcc, JCXZ не может выполнять дальних переходов. Проверка равенства CX нулю, например, может потребоваться в начале цикла, организованного командой LOOPNE, - если в него войти с CX = 0, то он будет выполнен 65 535 раз.
LOOP op1 - цикл
Уменьшает регистр CX на 1 и выполняет переход типа short на метку (которая не может быть дальше расстояния -128...+ 127 байт от команды loop), если CX не равен нулю. Эта команда используется для организации циклов, в которых регистр CX играет роль счетчика. Так, в следующем фрагменте команда ADD выполнится 10 раз:

```
mov cx, 0Ah
loop_start:
add ax,cx
loop loop_start
```

Команда LOOP полностью эквивалентна паре команд

```
dec ecx
jnz метка
```

Но LOOP короче этих двух команд на один байт и не изменяет значения флагов.
LOOPE op1 - цикл, пока равно
LOOPZ op1 - цикл, пока ноль
LOOPNE op1 - цикл, пока не равно
LOOPNZ - цикл, пока не ноль
Все перечисленные команды уменьшают регистр CX на один, после чего выполняют переход типа short, если ECX не равен нулю и если выполняется условие.
Для команд LOOPE и LOOPZ условием является равенство единице флага ZF, для команд LOOPNE и LOOPNZ - равенство флага ZF нулю. Сами команды LOOPcc не изменяют значений флагов, так что ZF должен быть установлен (или сброшен) предшествующей командой. Например, следующий фрагмент копирует строку из DS:SI в строку в ES:DI, пока не кончится строка (CX = 0) или пока не встретится символ с ASCII-кодом 13 (конец строки):

```
mov cx,str_length
move_loop:
lodsb
stosb
cmp al,13
loopnz move_loop
```

Десятичная арифметика и инструкции двоично-десятичной коррекции.
Вспомним про флаг AF - флаг полупереноса или вспомогательного переноса. Устанавливается в 1, если в результате предыдущей операции произошел перенос (или заем) из третьего бита в четвертый. Этот флаг используется автоматически командами двоично-десятичной коррекции.
Процессоры Intel поддерживают операции с двумя форматами десятичных чисел: неупакованное двоично-десятичное число - байт, принимающий значения от 00 до 09h, и упакованное двоично-десятичное число - байт, принимающий значения от 00 до 99h. Все обычные арифметические операции над такими числами приводят к неправильным результатам. Например, если увеличить 19h на 1, то получится число 1Ah, а не 20h. Для коррекции результатов арифметических действий над двоично-десятичными числами используются приведенные ниже команды.
DAA - коррекция после сложения

Если эта команда выполняется сразу после ADD (ADC, INC) и в регистре AL находится сумма двух упакованных двоично-десятичных чисел, то в AL записывается упакованное двоично-десятичное число, которое должно было стать результатом сложения. Например, если AL содержит число 19h, последовательность команд

inc al

daa

приведет к тому, что в AL окажется 20h (а не 1 Ah, как было бы после INC).

DAA выполняет следующие действия:

1. Если младшие четыре бита AL больше 9 или флаг AF = 1, то AL увеличивается на 6, CF устанавливается, если при этом сложении произошел перенос, и AF устанавливается в 1.
2. Иначе AF = 0.
3. Если теперь старшие четыре бита AL больше 9 или флаг CF = 1, то AL увеличивается на 60h и CF устанавливается в 1.
4. Иначе CF = 0.

Флаги AF и CF устанавливаются, если в ходе коррекции происходил перенос из первой или второй цифры. SF, ZF и PF устанавливаются в соответствии с результатом, флаг OF не определен.

DAS - коррекция после вычитания

Если эта команда выполняется сразу после SUB (SBB или DEC) и в регистре AL находится разность двух упакованных двоично-десятичных чисел, то в AL записывается упакованное двоично-десятичное число, которое должно было быть результатом вычитания. Например, если AL содержит число 20h, последовательность команд

dec al

das

приведет к тому, что в регистре окажется 19h (а не 1Fh, как было бы после DEC).

DAS выполняет следующие действия:

1. Если младшие четыре бита AL больше 9 или флаг AF = 1, то AL уменьшается на 6, CF устанавливается, если при этом вычитании произошел заем, и AF устанавливается в 1.
2. Иначе AF = 0.
3. Если теперь старшие четыре бита AL больше 9 или флаг CF — 1,то AL уменьшается на 60h и CF устанавливается в 1.
4. Иначе CF = 0.

Известный пример необычного использования этой команды - самый компактный вариант преобразования шестнадцатеричной цифры в ASCII-код соответствующего символа:

cmp al,10

sbb al,69h

das

После SBB числа 0-9 превращаются в 96h - 9Fh, а числа 0Ah - 0Fh - в 0A1h - 0A6h. Затем DAS вычитает 66h из первой группы чисел, переводя их в 30h - 39h, и 60h из второй группы чисел, переводя их в 41h — 46h.