Ташкентский Университет Информационных Технологий

# Тема: «Логические команды и команды сдвига»

Выполнила: Свитина Е.
Группа: 282-11 ХКТТр
Принял: Сейтназоров К.К.

# Логические команды и команды сдвига

- **Краткое описание группы логических команд**
- **Команды для выполнения логических операций**
- **Команды сдвига**
- **Организация работы с отдельными битами**

Наряду со средствами арифметических вычислений система команд процессора имеет также средства логического преобразования данных. Под ***логическим*** понимается такое преобразование данных, в основе которого лежат правила ***формальной логики.*** Формальная логика работает на уровне утверждений ***истинно*** или ***ложно.*** Для процессора это, как правило, означает 1 или 0 соответственно. Для компьютера язык нулей и единиц является родным, но минимальной единицей данных, с которой работают машинные команды, является байт. Однако на системном уровне часто необходимо иметь возможность работать на предельно низком уровне — на уровне битов.

К средствам логического преобразования данных в языке ассемблера относятся ***логические команды*** и ***логические операции*** (рис. 9.1). Команды рассматриваются в этой главе, операции были изучены нами в главе 5. Напомню, что операнд команды ассемблера в общем случае может представлять собой выражение, которое, в свою очередь, является комбинацией операторов и операндов. Среди этих операторов могут быть и операторы, реализующие логические операции над объектами выражения.

Перед знакомством с этими средствами давайте посмотрим, что же представляют собой сами логические данные и какие действия над ними могут производиться.

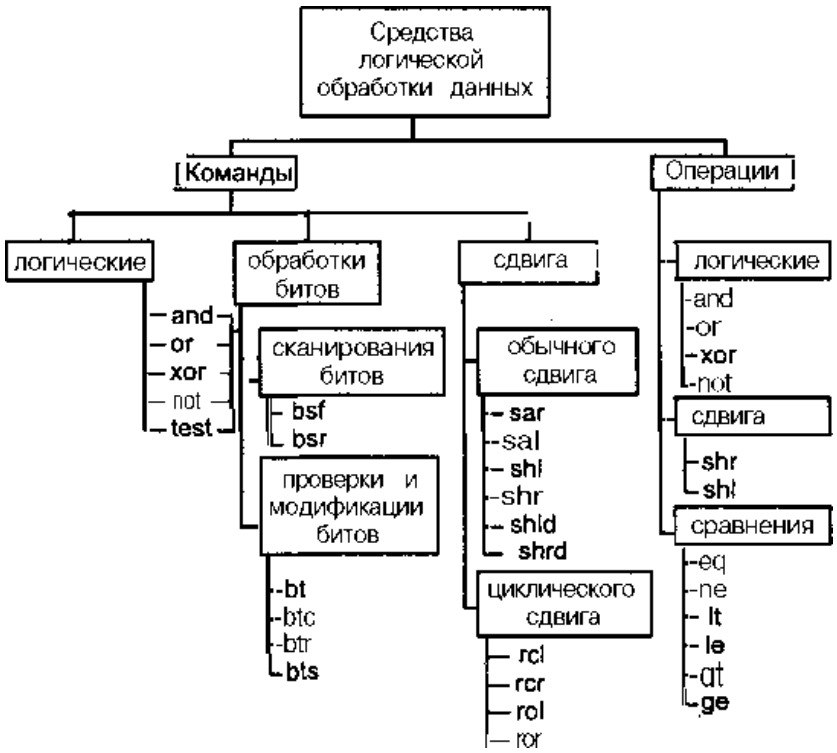


**Рис. 9.1.** Средства процессора для работы с логическими данными

# Логические данные

Теоретической базой для логической обработки данных является формальная логика. Существует несколько систем логики. Одна из наиболее известных — это ***исчисление высказываний. Высказывание*** — это любое утверждение, о котором можно сказать, что оно либо ***истинно,*** либо ***ложно.*** Исчисление высказываний представляет собой совокупность правил определения истинности или ложности некоторой комбинации высказываний.

Исчисление высказываний очень гармонично сочетается с принципами работы компьютера и основными методами его программирования. Все аппаратные компоненты компьютера построены на логических микросхемах. Система представления информации в компьютере на самом нижнем уровне основана на понятии ***бита.*** Бит, имея всего два состояния — 0 (ложно) и 1 (истинно), естественным образом вписывается в систему исчисления высказываний.

Согласно теории, над высказываниями (над битами) могут выполняться ***логические операции.***

- ***Отрицание*** (логическое ***НЕ)*** — логическая операция над одним операндом, результатом которой является величина, обратная значению исходного операнда. Эта операция однозначно

характеризуется следующей ***таблицей истинности***[I]**:**

| | | |
|---|---|---|
| Значение операнда: | 0 | 1 |
| Результат операции: | 1 | 0 |

- ***Логическое сложение*** (логическое ***включающее ИЛИ)*** **—** логическая операция над двумя операндами, результатом которой является истина (1), если один или оба операнда истинны (1), или ложь (0), если оба операнда ложны (0). Эта операция описывается с помощью следующей таблицы истинности:

| | | | | |
|---|---|---|---|---|
| Значение операнда 1: | 0 | 0 | 1 | 1 |
| Значение операнда 2: | 0 | 1 | 0 | 1 |
| Результат операции: | 0 | 1 | 1 | 1 |

- ***Логическое умножение*** (логическое ***И)*** **—** логическая операция над двумя операндами, результатом которой является истина (1) только в том случае, если оба операнда истинны (1). Во всех остальных случаях значение операции — ложь (0). Эта операция описывается с помощью следующей таблицы истинности:
-

| | | | | |
|---|---|---|---|---|
| Значение операнда 1: | 0 | 0 | 1 | 1 |
| Значение операнда 2: | 0 | 1 | 0 | 1 |
| Результат операции: | 0 | 0 | 0 | 1 |

is ***Логическое исключающее сложение*** (логическое ***исключающее ИЛИ)*** **—** логическая операция над двумя операндами, результатом которой является истина (1), если только один из двух операндов истинен (1), и ложь (0), если оба операнда либо ложны (0), либо истинны (1). Эта операция описывается с помощью следующей таблицы истинности:

| | | | | |
|---|---|---|---|---|
| Значение операнда 1: | 0 | 0 | 1 | 1 |
| Значение операнда 2: | 0 | 1 | 0 | 1 |
| Результат операции: | 0 | 1 | 1 | 0 |

# Логические команды

Система команд процессора содержит пять команд, поддерживающих описанные ранее операции. Эти команды выполняют логические операции над битами операндов. Размерность операндов, естественно, должна быть

[I] Таблица истинности — таблица результатов логических операций в зависимости от значений исходных операндов.

одинакова. Например, если размерность операндов равна слову (16 битов), то логическая операция выполняется сначала над нулевыми битами операндов, и ее результат записывается на место бита 0 результата. Далее команда последовательно повторяет эти действия над всеми битами с первого до пятнадцатого. Возможные варианты размерности операндов для каждой команды можно найти в приложении.

Далее перечислены базовые команды процессора, поддерживающие работу с логическими данными:

- and **операнд_1,операнд_2** — операция логического умножения. Команда выполняет поразрядно логическую операцию ***И (конъюнкцию)*** над битами операндов **операнд_1** и **операнд_2.** Результат записывается на место **операнд_1.**
- *or* **операнд_1,операнд_2** — операция логического сложения. Команда выполняет поразрядно логическую операцию ***ИЛИ (дизъюнкцию)*** над битами операндов **операнд_1** и **операнд_2.** Результат записывается на место **операнд_1.**
- *xor* **операнд_1,операнд_2** — операция логического исключающего сложения. Команда выполняет поразрядно логическую операцию исключающего ***ИЛИ*** над битами операндов **операнд_1** и **операнд_2.** Результат записывается на место **опе- ранд_1.**
- test **операнд_1,операнд_2** — операция проверки (способом логического умножения). Команда выполняет поразрядно логическую операцию ***И*** над битами операндов **операнд_1** и **операнд_2.** Состояние операндов остается прежним, изменяются только флаги ZF, SF, и PF, что дает возможность анализировать состояние отдельных битов операнда без изменения их состояния в исходных операндах. » not операнд — операция логического отрицания. Команда выполняет поразрядное инвертирование (замену значения на обратное) каждого бита операнда. Результат записывается на место операнда.

Для представления роли логических команд в системе команд процессора очень важно понять области их применения и типовые приемы их использования при программировании. Далее мы будем рассматривать логические команды в контексте обработки последовательности битов.

Очень часто некоторая ячейка памяти должна играть роль индикатора, показывая, например, занятость некоторого программного или аппаратного ресурса. Так как эта ячейка может принимать только два значения — занято (1) или свободно (0), то отводить под нее целый байт очень расточительно, логичнее для этой цели использовать бит. А если таких индикаторов много? Объединив их в пределах одного байта или слова, можно получить довольно

существенную экономию памяти. Посмотрим, что могут сделать для этого логические команды.

С помощью логических команд возможно выделение отдельных битов в операнде с целью их установки, сброса, инвертирования или просто проверки на определенное значение. Для организации подобной работы с битами второй операнд обычно играет роль ***маски.*** Путем установки в 1 битов этой маски и определяются нужные для конкретной операции биты первого операнда. Покажем, какие логические команды могут применяться для этой цели.

Для установки определенных разрядов (битов) в 1 применяется команда or **операнд_1,операнд_2**

В этой команде второй операнд, играющий роль маски, должен содержать единичные биты на месте тех разрядов, которые должны быть установлены в 1 в первом операнде:

*or eax.10b установить 1-й бит в регистре eax*

Для сброса определенных разрядов (битов) в 0 применяется команда

***and*** *операнд_1,операнд_2*

В этой команде второй операнд, играющий роль маски, должен содержать нулевые биты на месте тех разрядов, которые должны быть установлены в 0 в первом операнде:

*and* ***eax.*****0ffffffdh** *;сбросить в 0 1-й бит в регистре eax*

Для выяснения того, какие биты в обоих операндах различаются, или для инвертирования заданных битов в первом операнде применяется команда **xor операнд_1,операнд_2**

Интересующие нас биты маски (второго операнда) при выполнении команды XOR должны быть единичными, остальные — нулевыми:

```
xor eax,10b ;инвертировать 1-й бит в регистре eax
jz  mes     ;переход, если 1-й бит в al был единичным
```

Для проверки состояния заданных битов в первом операнде применяется команда

*test* ***операнд_1,операнд_2***

Проверяемые биты первого операнда в маске (втором операнде) должны иметь единичное значение. Алгоритм работы команды TEST подобен алгоритму работы команды AN D, но он не меняет значения первого операнда. Результатом команды является установка значения флага нуля ZF:

•если ZF = 0, то в результате логического умножения получился ненулевой результат, то есть хотя бы один единичный бит маски совпал с

соответствующим единичным битом первого операнда;

•если ZF= 1, то в результате логического умножения получился нулевой результат, то есть ни один единичный бит маски не совпал с соответствующим единичным битом первого операнда.

Таким образом, если любые соответствующие биты в обоих операндах установлены, то ZF = 0. Для реакции на результат команды TEST целесообразно использовать команду перехода на метку JNZ (Jump if Not Zero) — переход, если флаг нуля ZF ненулевой, или команду с обратным действием JZ (Jump if Zero) — переход, если флаг нуля ZF нулевой. Например,

```
testeax,00000010h
jnz m1   ;переход если 4-й бит равен 1
```

Начиная с системы команд процессора i386, набор команд для поразрядной обработки данных расширился. При использовании этих команд необходимо указывать одну из директив: .386, .486 и т. д. Следующие две команды позволяют осуществить поиск первого установленного в 1 бита операнда. Поиск можно произвести как с начала, так и от конца операнда:

•bsf операнд_1,операнд_2 — сканирование битов вперед (BitScaning Forward). Команда просматривает (сканирует) биты второго операнда от младшего к старшему (от бита 0 до старшего бита) в поисках первого бита, установленного в 1. Если таковой обнаруживается, в первый операнд заносится номер этого бита в виде целочисленного значения. Если все биты второго операнда равны 0, то флаг нуля ZF устанавливается в 1, в противном случае флаг ZF сбрасывается в 0.

```
mov al,02h
bsf bx,al   ;bx=l
jz  ml  ;переход, если al=00h
...
```

•bsr операнд_1,операнд_2 — сканирование битов в обратном порядке (Bit Scaning Reset). Команда просматривает (сканирует) биты второго операнда от старшего к младшему (от старшего бита к биту 0) в поисках первого бита, установленного в 1. Если таковой обнаруживается, в первый операнд заносится номер этого бита в виде целочисленного значения. При этом важно, что позиция первого единичного бита слева все равно отсчитывается относительно бита 0. Если все биты второго операнда равны 0, то флаг нуля ZF устанавливается в 1, в противном случае флаг ZF сбрасывается в 0.

Листинг 9.1 демонстрирует пример применения команд BSR и BSF. Введите код и исследуйте работу программы в отладчике (в частности,

обратите внимание на то, как меняется содержимое регистра BX после выполнения команд BSF и BSR).

**Листинг9.1. Сканирование битов**

```
;prg_9_1.asm
masm
model   small
stack   256
.data                   ;сегмент данных
.code                   ;сегмент кода
main:                   ;точка входа в программу
    mov ax,@data
    mov ds,ax
;...
.486                    ;это обязательно
    xor  ax,ax
    mov al,02h
    bsf  bx,ax          ;bx=l
    jz   ml             ;переход,  если  al=00h
    bsr  bx,ax
ml:
;...
    mov ax,4c00h;стандартный выход
    int 21h
end main
```

Интерес представляют еще несколько из группы логических команд, позволяющих реализовать доступ к конкретному биту операнда. Они, как и предыдущие, появились в моделях процессоров Intel, начиная с i386. Поэтому при их использовании не забывайте указывать одну из директив: .386, .486 и т. д. Операнд может находиться как в памяти, так и в регистре общего назначения. Положение бита задается смещением его относительно младшего бита операнда. Смещение может как задаваться в виде непосредственного значения, так и содержаться в регистре общего назначения. В качестве значения смещения вы можете использовать результаты работы команд BSR и BSF. Все команды присваивают значение выбранного бита флагу CF:

```
bt  операнд,смещение_бита
```

Например,

```
bt  ax,5     ;проверить  значение  бита  5
jnc ml       ;переход, если бит = 0
```

Команда проверки и установки бита BTS (Bit Test and Set) переносит значение бита в флаг CF и затем устанавливает проверяемый бит в 1:

```
bts  операнд,смещение_бита
```

Например,

```
mov ax,10
bts pole,ax ;проверить и установить 10-й бит в pole
jc  ml      ;переход,  если проверяемый бит был равен 1
```

Команда проверки и инвертирования бита BTC (Bit Test and Convert) переносит значение бита в флаг CF и затем инвертирует значение этого бита:

btc **операнд,смещение_бита**

**Команды сдвига**

Команды сдвига также обеспечивают манипуляции над отдельными

битами операндов, но иным способом, чем рассмотренные ранее логические команды. Все команды сдвига перемещают биты в поле операнда влево или вправо, в зависимости от кода операции. Все команды сдвига имеют одинаковую структуру: коп **операнд, счетчик-сдвигов**

Количество сдвигаемых разрядов (значение счетчик-сдвигов) может задаваться двумя способами:

II статически — непосредственно во втором операнде;

•динамически — в регистре CL перед выполнением команды сдвига.

Исходя из размерности регистра CL, понятно, что значение счетчик-сдвигов может лежать в диапазоне от 0 до 255. Но на самом деле это не совсем так. В целях оптимизации процессор воспринимает только значения пяти младших битов счетчика, то есть значение лежит в диапазоне от 0 до 31. В последних моделях процессора есть дополнительные команды, позволяющие делать 64-разрядные сдвиги. Мы их рассмотрим чуть позже.

Все команды сдвига устанавливают флаг переноса CF. По мере сдвига битов за пределы операнда они сначала попадают во флаг переноса CF, устанавливая его равным значению очередного бита, оказавшегося за пределами операнда. Куда этот бит попадет дальше, зависит от типа команды сдвига и алгоритма программы.

По принципу действия команды сдвига можно разделить на два типа:

И команды ***линейного сдвига;***

***к*** команды ***циклического сдвига.***

**Линейный сдвиг**

К командам линейного сдвига относятся команды, осуществляющие сдвиг по следующему алгоритму.

1. Очередной «выдвигаемый» бит устанавливает **флаг** CF.
2. Бит, появляющийся с другого конца операнда, имеет значение 0.
3. При сдвиге очередного бита он переходит во флаг CF, при этом значение предыдущего сдвинутого бита ***теряется***!

Команды линейного сдвига делятся на два подтипа: ti команды ***логического*** линейного сдвига; ***ш*** команды ***арифметического*** линейного сдвига.

Далее перечислены команды логического линейного сдвига:

*Ш* **shl операнд,счетчик_сдвигов** — логический сдвиг влево (Shift Logical Left). Содержимое операнда сдвигается влево на количество битов, определяемое значением **счетчик_сдвигов.** Справа в позицию младшего бита вписываются нули.

shr **операнд,счетчик_сдвигов** — логический сдвиг вправо (Shift Logical Right). Содержимое операнда сдвигается вправо на количество битов, определяемое значением **счетчик_сдвигов.** Слева в позицию старшего (знакового) бита вписываются нули.

Рисунок 9.2 иллюстрирует принцип работы этих команд.

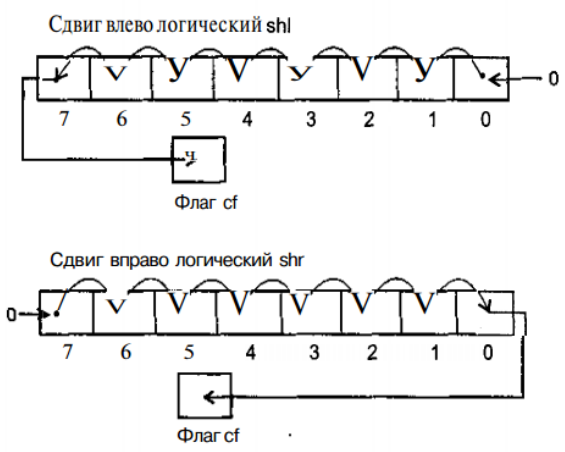


**Рис. 9.2.** Схема работы команд линейного логического сдвига

Ниже показан фрагмент программы, выполняющей преобразование двух неупакованных BCD-чисел в слове памяти bcd_dig в упакованное BCD-число в ре- гистреAL.

```
...
bcd_dig dw  0905h    ;описание неупакованного BCD-числа 95
...
    mov ax,bcd_dig   ;пересылка
    shl ah,4         ;сдвиг влево
    add al,ah        ;сложение для получения результата: al=95h
```

.

Команды арифметического линейного сдвига отличаются от команд логического сдвига тем, что они особым образом работают со знаковым разрядом операнда:

ii **5a1операнд,счетчик_сдвигов** — арифметический сдвиг влево (Shift Arithmetic Left). Содержимое операнда сдвигается влево на количество битов, определяемое значением **счетчик_сдвигов.** Справа (в позицию младшего бита) вписываются нули. Команда SAL не сохраняет знака, но устанавливает флаг OF в случае смены знака очередным выдвигаемым битом. В остальном команда SAL полностью аналогична команде SHL;

*ш* sar операнд,счетчик_сдвигов — арифметический сдвиг вправо (Shift Arithmetic Right). Содержимое операнда сдвигается вправо на количество битов, определяемое значением счетчик_сдвигов. Слева в операнд вписываются нули. Команда SAR сохраняет знак, восстанавливая его после сдвига каждого очередного бита.

Рисунок 9.3 иллюстрирует принцип работы команд линейного арифметического сдвига.

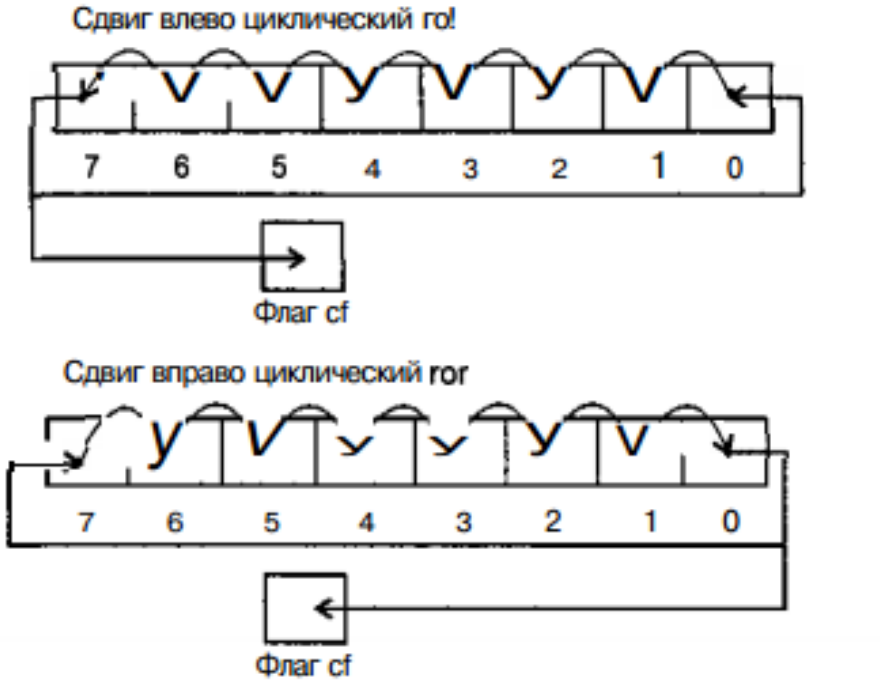


**Рис. 9.4.** Схема работы команд простого циклического сдвига

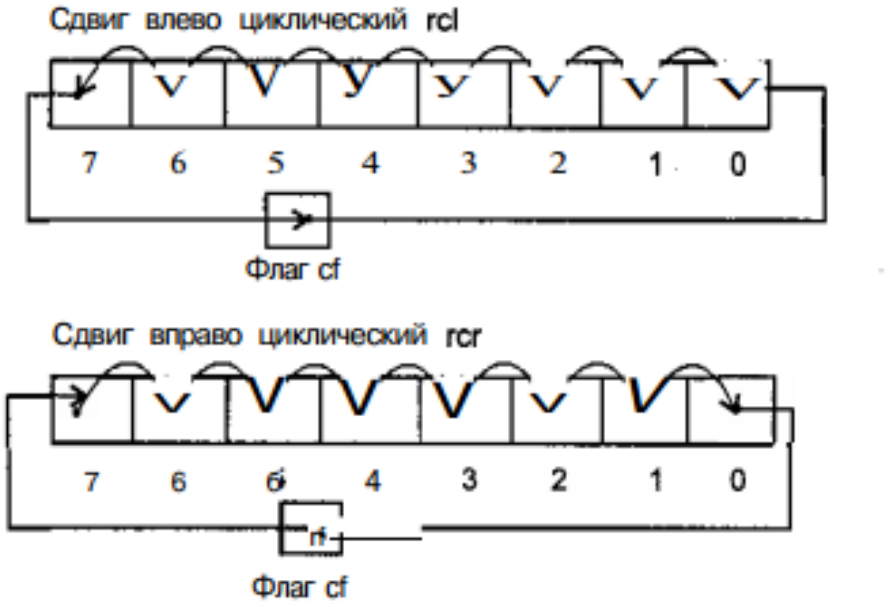


**Рис. 9.5.** Схема работы команд циклического сдвига через флаг перенос

**Циклический сдвиг**

К командам циклического сдвига относятся команды, сохраняющие значения сдвигаемых битов. Есть два типа команд циклического сдвига:

' - команды простого циклического сдвига (рис. 9.4);

•команды циклического сдвига через флаг переноса CF (рис. 9.5).

Далее перечислены команды простого циклического сдвига:

:Ш **rot операнд,счетчик-сдвигов** — циклический сдвиг влево (Rotate Left). Содержимое операнда сдвигается влево на количество битов,

определяемое операндом **счетчик-сдвигов.** Сдвигаемые влево биты записываются в тот же операнд справа.

•**гог операнд,счетчик-сдвигов** — циклический сдвиг вправо (Rotate Right). Содержимое операнда сдвигается вправо на количество битов, определяемое операндом счетчик_сдвигов. Сдвигаемые вправо биты записываются в тот же операнд сл. Как видно из рис. 9.4, команды простого циклического сдвига в процессе своей работы осуществляют одно полезное действие: циклически сдвигаемый бит не только вдвигается в операнд с другого конца, но и одновременно его значение становится значением флага CF. К примеру, для того чтобы обменять содержимое двух половинок регистра EAX, достаточно выполнить следующую последовательность команд:

```
mov eax,ffff0000h
mov cl,16
rol eax,cl
```

Команды циклического сдвига через флаг переноса CF отличаются от команд простого циклического сдвига тем, что сдвигаемый бит не сразу попадает в операнд с другого его конца, а сначала записывается во флаг переноса CF. Лишь следующее исполнение данной команды сдвига (при условии, что она выполняется в цикле) приводит к помещению выдвинутого ранее бита в другой конец операнда (рис. 9.5). Команды циклического сдвига через флаг переноса CF перечислены далее:

rcl **операнд,счетчик_сдвигов** — циклический сдвиг влево через перенос (Rotate through Carry Left). Содержимое операнда сдвигается влево на количество битов, определяемое операндом **счетчик_сдвигов.** Сдвигаемые биты поочередно становятся значением флага переноса **CF;**

rcr **операнд,счетчик_сдвигов** — циклический сдвиг вправо через перенос (Rotate through Carry Right). Содержимое операнда сдвигается вправо на количество битов, определяемое операндом **счетчик_сдвигов.** Сдвигаемые биты поочередно становятся значением флага переноса CF.

Из рис. 9.5 видно, что при сдвиге через флаг переноса появляется промежуточный элемент, с помощью которого можно производить подмену циклически сдвигаемых битов, в частности, выполнять рассогласование битовых последовательностей. Под рассогласованием битовой последовательности здесь и далее подразумевается действие, которое позволяет некоторым образом локализовать и извлечь нужные участки этой последовательности и записать их в другое место. Например, рассмотрим, как переписать в регистр BX старшую половину регистра EAX с одно-временным ее обнулением в регистре EAX:

```
    mov cx,16       ;кол-во сдвигов для eax
m1:
    clc              ;сброс флага cf в 0
    rcl  eax,1      ;сдвиг крайнего левого бита из eax в cf
    rcl bx,1         ;перемещение бита из cf справа в bx
    loopm1          ;цикл 16 раз
    rol eax,16      ;восстановить правую часть eax
```

Команды простого циклического сдвига можно использовать для операций другого рода. К примеру, подсчитаем количество единичных битов в регистре EAX:

```
    xor dx,dx       ;очистка dx для подсчета единичных битов
    mov cx,32        ;число циклов подсчета
cycl:               ;метка цикла
    ror eax,1        ;циклический сдвиг вправо на 1 бит
    jnc  not_one     ;переход, если очередной бит в cf
                     ;не равен единице
    inc  dx         ;увеличение счетчика цикла
not_one:
    loopcycl        ;переход на метку cycl, если
                     ;значение в cx не равно 0
```

Этот фрагмент не требует особых пояснений, единственное, что нужно помнить, — особенности работы команды цикла ШОР (глава 10). Команда ШОР сравнивает значение регистра ECX/CX с нулем и, если оно не равно нулю, выполняет уменьшение ECX/CX на единицу и передает управление на метку в программе, указанную в этой команде в качестве операнда.

**Дополнительные команды сдвига**

Система команд моделей микропроцессоров Intel, начиная с 80386, содержит дополнительные команды сдвига, расширяющие рассмотренные нами ранее возможности. Это — ***команды сдвига двойной точности:***

**shld операнд_1,операнд_2,счетчик_сдвигов** — сдвиг влево двойной точности. Команда сдвигает биты первого операнда влево и заполняет его справа значениями битов, вытесняемых из второго операнда, согласно схеме на рис. 9.6. Количество сдвигаемых битов определяется значением счетчика сдвигов, которое может лежать в диапазоне 0...31. Это значение может задаваться непосредственно в третьем операнде или содержаться в регистре CL. Значение второго операнда не меняется;


**Рис. 9.6.** Схема работы команды SHLD

is **shrd операнд_1,операнд_2,счетчик_сдвигов** — сдвиг вправо двойной точности. Команда сдвигает биты первого операнда вправо и заполняет его слева значениями битов, вытесняемых из второго операнда, согласно схеме на рис. 9.7. Количество сдвигаемых битов определяется значением счетчика сдвигов, которое может лежать в диапазоне 0...31. Это значение может задаваться непосредственно в третьем операнде или содержаться в регистре CL. Значение второго операнда не меняется.


**Рис. 9.7.** Схема работы команды SHRD

Как мы отметили, команды SHLD и SHRD осуществляют сдвиги на величину до 32 разрядов, но за счет особенностей задания операндов и алгоритма работы эти команды можно использовать для работы с полями длиной до 64 битов. Например, рассмотрим, как можно осуществить сдвиг влево на 16 битов поля из 64 битов.

```
;....
.data
pole_l  dd  0b21187f5h
pole_h  dd  45ff6711h
.code
;...
.386
    mov cl,16   ;загрузка счетчика сдвигов в cl
    mov eax,pole_h
    shldpole_l,eax,cl
    shl pole_h,cl   ;pole_l=87f50000h;
pole_h=6711b211h
```

Рассмотрим еще некоторые наиболее типичные примеры применения этих команд. Отметим следующий момент. Рассмотренные далее действия, конечно, можно выполнить и множеством других способов, но эти являются самыми быстрыми. Если ваши программы должны работать максимально быстро, то есть смысл потратить время на разбор этих примеров.

**Примеры работы с битовыми строками. Рассогласование битовых строк**

Наглядный пример рассогласования последовательностей битов — преобразование неупакованного BCD-числа в упакованное BCD-число. Один из вариантов такого преобразования был рассмотрен нами ранее при обсуждении команды линейного сдвига SHL. Попробуем выполнить подобное преобразование с использованием команд сдвига двойной точности (листинг 9.2). В общем случае длина числа может быть произвольной, но при этом нужно учитывать ограничения, которые накладываются используемыми ресурсами процессора. Ограничения связаны в основном с тем, что центральное место в преобразовании занимает регистр EAX, поэтому, если преобразуемое число имеет размер более четырех байтов, то его придется делить на части. Но это уже чисто алгоритмическая задача, поэтому в нашем случае предполагается, что неупакованное BCD-число имеет длину 4 байта.

**Листинг 9.2.** Преобразование BCD-числа (вариант 2)

```
;prg_9_2.asm
masm
model    small
stack    256
.data
len=4                       ;длина неупакованного BCD-числа
unpck_BCD    label    dword
dig_BCD  db  2,4,3,6        ;неупакованное BCD-число 6342
pck_BCD  dd  0              ;pck_BCD=00006342
.code
main:                       ;точка входа в программу
    mov  ax,@data
    mov  ds,ax
    xor  ax,ax
    mov  cx,len
.386                        ;это обязательно
    mov  eax,unpck_BCD
m1:
    shl  eax,4              ;убираем нулевую тетраду
    shld pck_BCD,eax,4      ;тетраду с цифрой заносим в поле pck_BCD
    shl  eax,4              ;убираем тетраду с цифрой из eax
    loop m1                 ;цикл
    exit:                    ;pck_BCD=00006342
    mov  ax,4c00h
    int  21h
end      main
```

Команды сдвига двойной точности SHLD и SHRD позволяют осуществлять с максимально возможной скоростью вставку битовой строки из регистра в произвольное место другой (большей) строки битов в памяти и извлечение в регистр битовой подстроки из некоторой строки битов в памяти. В результате этих операций смежные с подстрокой биты по ее обеим сторонам остаются неизменными.

**Вставка битовых строк**

Рассмотрим пример вставки битовой строки длиной 16 битов, находящейся в регистре EAX, в строку памяти str, начиная с ее бита 8 (листинг 9.3). Вставляемая битовая строка выровнена к левому краю регистра EAX.

**Листинг 9.3.** Вставка битовой строки

```
<1>     ;prg_9_3.asm
<2>     raasm
<3>     model    small
<4>     stack    256
<5>     .data
<6>     bit_str dd  11010111h   ;строка для вставки
<7>     P_str   dd  0ffff0000h  ;вставляемая подстрока 0ffffh
<8>     .code
<9>     main:                   ;точка входа в программу
<10>        mov ax,@data
<11>        mov ds,ax
<12>        xor ax,ax
<13>    .386                    ;это обязательно
<14>        mov eax,p_str
<15>    ;правый край места вставки циклически переместить к краю
<16>    ;строки bit_str (сохранение правого контекста):
<17>        ror bit_str,8
<18>        shr bit_str,16      ;сдвинуть строку вправо
                                ;на длину подстроки (16 битов)
<19>        shld bit_str,eax,16 ;сдвинуть 16 бит
<20>        rol bit_str,8       ;восстановить младшие 8 бит
<21>    ;...
<22>        exit:               ;bit_str=11ffff11
<23>        mov ax,4c00h
<24>         int 21h
<25>    end main
```

Листинг 9.3 удобно исследовать в отладчике. При этом важно понять зависимость между непосредственными значениями, используемыми в

командах строк 17-20, и исходными значениями. Общая методика вставки битовых строк выглядит следующим образом.

1. Подогнать к правому краю строки младший бит места вставки в этой строке. Делать это нужно командой циклического сдвига, чтобы сохранить правую часть исходной строки. Величина сдвига определяется очень просто — это номер начальной позиции места вставки (строка 17).
2. Сдвинуть исходную строку вправо на количество битов, равное длине вставляемой подстроки (строка 18). Эти биты нам больше не нужны, поэтому для сдвига используется команда простого сдвига **SHR,**
3. Командой SH LD вставить вставляемую подстроку в исходную подстроку. Перед этим, естественно, левый край вставляемой подстроки находится у левого края регистра EAX (строка 19).
4. Восстановить командой циклического сдвига правую часть исходной строки (строка 20).

Наибольшей эффективности при использовании этой программы можно достичь, если оформить представленную в ней последовательность команд в виде макрокоманды. Понятие макрокоманды будет рассматриваться нами в главе 14, но сейчас важно отметить, что в данном случае она позволит нам не задумываться о настройке строк 17-20 на конкретную вставку. При изучении материала главы 14 вы можете поэкспериментировать с данной программой, разработав на ее основе макрокоманду.

**Извлечение битовых строк**

Рассмотрим пример извлечения 16 битов из строки в памяти **bit_str,** начиная с бита 8, в регистр EAX (листинг 9.4). Результат следует выровнять по правому краю регистра EAX; строка bit_str не изменяется. Этот пример можно рассматривать как обратный тому, который мы только что привели в листинге 9.3. Методика извлечения битовой подстроки, если вы разобрались с программой вставки битовой строки, не должна вызвать у вас затруднений.

**Листинг 9.4.** Извлечение битовой строки

```
;prg_9_4.asm
masm
model   small
stack   256
.data
bit_str dd  11ffff11h   ;строка для извлечения
.code
main:                   ;точка входа в программу
    mov ax,@data
    mov ds,ax
    xor ax,ax
.386                    ;это обязательно
;левый край места извлечения циклически переместить к левому краю
;строки bit_str  (сохранение левого контекста)
    rol bit_str,8
    mov ebx,bit_str     ;подготовленную строку в ebx
    shld eax,ebx,16      ;вставить извлекаемые 16 бит
    ;в регистр  eax
    ror bit_str,8       ;восстановить старшие 8 бит
;...
    exit:               ;eax=0000ffff
    mov ax,4c00h
    int 21h
end  main
```

**Пересылка битов**

По сути, программа пересылки битов является комбинацией двух предыдущих. Поэтому попробуйте самостоятельно разработать программу пересылки блока битов из одной битовой строки в другую, взяв за основу только что рассмотренные программы (см. листинги 9.3 и 9.4). К примеру, пусть имеется две битовые строки:

```
bit_str1    dd  0abcdefabh
bit_str2    dd  012345678h
```

Из этих строк получите строку

```
bit_str2   dd  0abcd34abh
```

**Итоги**

Минимально адресуемая единица данных в процессоре — байт. Логические команды позволяют манипулировать отдельными битами. Только эти команды . в системе команд процессора позволяют работать на битовом уровне. Этим, в частности, объясняется их важность.

Работа на битовом уровне позволяет в отдельных случаях существенно сэкономить память, особенно при моделировании различных массивов, содержащих одноразрядные флаги или переключатели.

Команды сдвига позволяют выполнять быстрое умножение и деление операндов на степени двойки, а также эффективное преобразование данных.

Применение команд циклического сдвига и сдвига двойной точности позволяет реализовать максимально быстрые операции по рассогласованию, перемещению, вставке и извлечению битовых подстрок.