# **8. Логические команды** Логические команды выполняют над операндами логические (побитовые) операции, то есть они рассматривают коды операндов не как единое число, а как набор отдельных битов. Этим они отличаются от арифмети-ческих команд. Логические команды выполняют следующие основные операции:

#     логическое И - AND    логическое ИЛИ – OR   исключающее ИЛИ - XOR;  NOT инвертирует бит источника  логические сдвиги SHR SHL  арифметические сдвиги SAL SAR  циклические сдвиги ROR ROL RCR RCL   проверка битов и операндов;   установка и очистка битов (флагов)  регистра состояния процессора ( PSW).

# **8.1.Побитовые логические команды** NOT (~) устанавливает обpатное значение бит, нули ,будут  единицами,  а единицы – нулями. AND (&): Если оба из сравниваемых битов равны 1, то результат равен 1; иначе = 0. OR (|): Если хотя бы один из сравниваемых битов  равен  1,  то  результат = 1; если сравниваемые биты равны 0, то результат =0. XOR (^): Если один из битов =0, а дру-гой=1,  то результат=1; если сравниваемые биты одинаковы (оба - 0 или оба - 1)то результат=0. TEST: действует как AND-устанавливает флаги, но не изменяет биты.

# Пример 1. int a,b,c; \_asm { mov ecx, 100b mov ebx, 100b mov ecx, 100b and eax, 100 or ebx, 100 xor ecx,100 mov a,eax mov b,ebx mov c,ecx } cout<<a<<‘ ’<<b<<‘ ’<<c; 68 100 96

# Пример 2: mov ax, 3406 mov dx, 13EAh xor ax, dx ax = 340610, в двоичном - 000011010100111010 . dx = 13EA 16, в двоичном - 0001001111101010. Выполнение операции XOR на этими битами: Источник = 00010011111010102 (dx) Приемник = 00001101010011112 (ax) Результат = 00011110101001012 (значение в ax) 00011110101001012 =784510 =1EA516.

# Доказать, что с помощью следующих трех команд можно обменять содержимое двух регистров, например, AX и BX : XOR AX,BX XOR BX,AX XOR AX,BX

# **8.2. Команды сдвигов.** Команды сдвигов позволяют побитно сдвигать код операнда вправо (в сторону младших разрядов) или влево (в сторону старших разрядов). Тип сдвига (логический, арифметический или циклический) определяет, каково будет новое значение старшего бита (при сдвиге вправо) или младшего бита (при сдвиге влево), а также определяет, будет ли где-то сохранено прежнее значение старшего бита (при сдвиге влево) или младшего бита (при сдвиге вправо).

# Команды используются для сдвига разрядов операнда влево. Их машинные коды идентичны. Значение второго операнда (счетчик сдвига) ограничено диапазоном 0...31,т.к. микро-процессор использует только пять младших разрядов операнда количество\_разрядов. Во всех командах второй операнд можно задать числом или использовать регистр CL. *При использовании других регистров будут ошибки*. .

# Аналогично другим командам сдвига сохраняется эффект, связанный с поведением флага of, значение которого имеет смысл только в операциях сдвига на один разряд: если of=1, то текущее значение флага cf и выдвигаемого слева бита операнда различны; если of=0, то текущее значение флага cf и выдвигаемого слева бита операнда совпадают. Этот эффект обусловлен тем, что флаг of устанавливается в единицу всякий раз при изменении знакового разряда операнда.

# **8.2.1 Сдвиг арифметический влево** SAL операнд, количество\_сдвигов ⮚сдвиг всех битов операнда влево на один разряд, выдвигаемый слева бит помещается значением флага переноса cf; ⮚ одновременно справа в операнд вдвигается нулевой бит; ⮚ указанные выше два действия повторяются количество раз, равное значению второго операнда.

# **8.2.2 Сдвиг логический операнда влево** SНL операнд,количество\_сдвигов  Сдвиг логический операнда влево ⮚сдвиг всех битов операнда влево на один разряд, выдвигаемый слева бит помещается значением флага переноса cf; ⮚ одновременно справа в операнд вдвигается нулевой бит; ⮚ указанные выше два действия повторяются количество раз, равное значению второго операнда.

# Команды удобно использовать для умножения целочисленных операндов без знака на 2. Самый быстрый способ умножения;  умножить содержимое ax на 16 (24).      mov   ax,17         shl     ax,4

# **8.2.3 Сдвиг арифметический вправо** SAR операнд,количество\_сдвигов ⮚сдвиг всех битов операнда вправо на один разряд, выдвигаемый справа бит становится значением флага переноса cf; ⮚ одновременно слева в операнд вдвигается значение старшего бита операнда, т. е. по мере сдвига вправо освобождающиеся места заполняются значением знакового разряда ; ⮚ указанные действия повторяются количество раз, равное значению 2 операнда.

# **8.2.4 Сдвиг логический вправо** SHR операнд, кол-во\_сдвигов ⮚ сдвиг всех битов операнда вправо на один разряд; ⮚ при этом выдвигаемый справа бит становится значением флага переноса cf; одновременно слева в операнд вдвигается нулевой бит; ⮚ указанные выше два действия повторяются количество раз, равное значению второго операнда.

# Команды используются для логического сдвига разрядов операнда вправо. Значение второго операнда (счетчика сдвига) ограничено диапазоном 0...31. Это объясняется тем, что микропроцессор использует только пять младших разрядов операнда количество разрядов. Флаг of всегда сбрасывается в ноль в операциях сдвига на один разряд. Команды можно использовать для деления целочисленных операндов без знака на 2n.

# Операнд количество\_разрядов можно задать константой или записать в регистр cl. mov     cl,4         shr     eax,cl  ;24

# Пример 1. mov AL,7 sal AL,1 //AL= 0Eh=7\*2, CF=0 mov AX,1Fh mov CL,8 sal AX,CL //AX=lF00h=lFFh\*256, CF=0 mov SI,-1 //SI=FFFFh mov CL,4 sal SI,CL //SI=FFF0h=-l\*16=-16, CF=1 mov EBX,0000C835h sal EBX,5 //EBX=001906A0h

# Пример 2. mov AL,7 sar AL,1 ;AL=3=7/2, CF=1. Остаток потерян mov AX,lFF0h mov CL,4 sar AX,CL //;AX=01FFh=lFF0h/16, CF=0 mov BX,-8 ;BX=FFF8h mov CL,2 sar BX,CL ;BX=FFFEh=-2=-8/4, CF=0 mov EAX,0F0001234h sar EAX,8 ;EAX=FFF00012h

# **8.3 Циклические сдвиги** Циклические сдвиги позволяют сдвигать биты кода операнда по кругу (по часовой стрелке при сдвиге вправо или против часовой стрелки при сдвиге влево). При этом в кольцо сдвига может входить или не входить флаг переноса. В бит флага переноса (если он используется) записывается значение старшего бита при циклическом сдвиге влево и младшего бита при циклическом сдвиге вправо. Соответственно, значение бита флага переноса будет переписываться в младший разряд при циклическом сдвиге влево и в старший разряд при циклическом сдвиге вправо.

# **8.3.1 Циклический сдвиг операнда влево через флаг переноса cf** RCL операнд,количество\_сдвигов ⮚ сдвиг всех битов операнда влево на один разряд, при этом старший бит операнда становится значением флага переноса cf; ⮚ старое значение флага переноса cf вдвигается в операнд справа и становится значением младшего бита операнда; ⮚ указанные выше два действия повторяются количество раз, равное значению второго операнда команды rcl.

# clc //Сбросим флаг CF mov EAX, 2 rcr EАХ,1 //EАХ=1, CF=0 stc //Установим флаг CF mov DL,8 rcr DL,1 //DL=84h, CF=0

# Команды установки и очистки битов регистра состояния процессора (то есть флагов) позволяют установить или очистить любой флаг, что бывает очень удобно. Каждому флагу обычно соответствуют две команды, одна из которых устанавливает его в единицу, а другая сбрасывает в нуль. Флагу переноса C (от Carry) будут соответствовать команды CLC (очистка) и SEC или STC (установка).

# **8.4 Запись содержимого регистра АН в регистр флагов SAHF** Команда sahf копирует разряды 7, 6, 4, 2, 0 регистра АН в регистр флагов процессора, устанавливает значения флагов SF, ZF, AF, PF,CF соответственно. Команда без операндов. Команда sahf позволяет изменять значения этих флагов процессора, которые нельзя изменить непосредственно. Но команда sahf заполняет только младший байт регистра флагов. Поэтому нельзя изменить с ее помощью, например, состояние флага OF.

# lahf ;Регистр АН отображает ;состояние регистра флагов or АН,80h Установка бита 7 = SF sahf ;Загрузка АН в регистр флагов, где теперь будет SF=1

# Пример 5. Программа возведения числа в квадрат. 25 10=19 16 = 11001 2 11001 \* 11001 11001 + 00000 00000 11001 11001 10011100012

# mov dx,x xor ax,ax mov bx,x nn: shr bx,1 ;множитель = 0? jnc ne add ax,dx ne: shl dx,1 ;получение множителя cmp bx,0 ;все множители? jne nn kon:

# Пример 6. Программа умножения Х на 10 с использованием логических операций. \_asm { mov eax, х // х=5 shl eax,3 // eaх=5\*2\*2\*2=40 mov ebx,x shl ebx, 1 // ebх=5\*2=10 add eax, ebx mov y,eax }

# Для умножение на 2 нужно сдвинуть регистр влево на 1 бит например командой SHR, старший бит уйдет в CF, чтобы его не потерять сдвинем второй регистр командой RCL: SHL AX,1; RCL DX,1 //= DX:AX\*2

# Умножение на 4 соответсвенно два раза: SHL AX,1; RCL DX,1; SHL AX,1; RCL DX,1 //= DX:AX\*4 Деление - сдвиг вправо: SHR DX,1; RCR AX,1 ///= DX:AX/2 Ну и отсюда следует: SHL BX,1; RCL AX,1 RCL DX,1 /// = DX:AX:BX\*2

# Пример 7. Определение разрядности целого числа. Простейший способ сдвигать беззнаковое целое вправо, пока оно не станет равным 0. Количество сдвигов и есть его разрядность.   int IntLen(int x) { int i; for (int i=0; x!=0; i++, x>>=1); return i; }

# Определение разрядности целого числа. int IntLen(int x) { int i; \_asm { xor eax,eax loop1: cmp x,0 je end inc eax shr x,1 jmp loop1 end:mov i,eax } return i; }

# Пример 8. Подсчет количества единичных битов. При сдвиге вправо все биты числа будут последовательно находиться в младшем раз-ряде, из которого их нужно выделять с исполь-зованием операции И с единичной маской. int what\_is\_1(int n) { int i,s; for (i=0,s=0; i < sizeof(int) \* 8; i++) { if (n & 1) s++; n >>=1; } return s; }

# \_asm { mov ecx,4\*8 mov s,0 xor eax,eax loop1: shr n,1 jnc else\_ inc s else\_: loop loop1 } return s; }

# Пример 9. Переворачивание двоичного числа int revers\_bit(int n) // CИ- самостоятельно { int i; \_asm { xor ebx,ebx loop1: xor eax,eax clc

# cmp n,0 je end shr n,1 rcl eax,1 shl ebx,1 add ebx,eax jmp loop1 end: mov r,ebx } return r; }

# Пример 10. Печать в 16 с/с десятичного числа  void dump16с\_с(int n) // { int i; char s[17]; \_asm { xor esi,esi mov ecx,8 loop1: xor eax,eax mov eax,n

# and eax,0f0000000h shr eax,28 add al,30h mov byte ptr s[esi],al inc esi shl n,4 loop loop1 end: mov s[esi],0 mov i,ebx } printf("%s",s); }

# Пример 11. Преобразование натурального числа, находящегося в переменной типа Int, в представление типа Double   Решение. 1. -256.5 == - 100.816 2.   - 100.8 == - 1 0000 0000 . 10002 3.   - 1 0000 0000 . 1000 == - 1 . 0000 0000 1000 \* 2 4.   h =7F + 8 = 8716== 1000 0111 2 5. В первом бите четырехбайтового числа 1.

# 6. В следующих восьми битах записывается характеристика( 1000 0111 ) 7. В оставшиеся биты записываем усеченную мантиссу С3 80 40 00. 8. Перевернув, 00 40 80 С3. Восьмибайтовое под характеристику отводится 11 битов , а сама h == 3FF + р , где р - усеченный порядок числа с плавающей точкой в 2 с/с, да еще тем, что под усеченную мантиссу отводится 52 бита

# #include <stdio.h> #include <string.h> double conv( int x) { double y; \_asm { lea edi,y xor ebx,ebx mov ecx,8 mov eax,x cdq

# // {количество незнач нулей} nn1: shl eax,1 rcl edx,1 inc ebx jnc nn1 //{мантиссу вправо на свое место} mov ecx,12 nn2: shr edx,1 rcr eax,1 loop nn2

# //{ характеристику на свое место в регистре} mov esi,64 sub esi,ebx add esi,03ffh mov cl,20 //{характеристика в 2-х байтах} shl esi,cl // число с плав. точкой EDX:EAX // в EAX уже есть младшая часть мантиссы // к EDX добавим характеристику}

# Пример 12 Определить является ли заданное натуральное число простым. int Prost( int x) { mov r,1 mov eax,x //x->eax xor edx,edx //очистить edx cmp eax,1 //сравнить eax c 1 je \_exit //если eaх=1 то перейти на \_exit mov ecx,2 //иначе ecx:=2

# \_while: cmp ecx,y // сравнить ecx c y jg \_exit1 //если ecx>y то перейти на \_exit1 xor edx,edx //очистить edx mov eax,x //x->eax div ecx //делим edx:eax на ecx }

# void main() { int k,a; cout<<"a="; cin>>a; k=Prost(a); if (k>0) cout<<"Prostoe"; else cout<<"Sostavnoe"; }

# Повторение Битовые операторы С++

## Пример 13. Сосчитать количество единиц в двоичной записи числа *i*, заданного в десятичной системе счисления.

## int bit;

## int mask = 1; //двоичное число 0000 0000 0000 0000 0000 0000 0000 0001 int num = 0; for ( int i = 0; i <=31; i++) { //проверка битовой 1 в соответствующем разряде} bit=mask & a; if (bit == mask) num++; mask = mask << 1 ; }

## Если число *i* является степенью двойки, то

## *i* & (*i*– 1) = 0.

## Аналогично, если в двоичной записи числа все 1, то

## *i* & (*i*+ 1) = 0.

## Обнулить крайний правый единичный бит числа A

## Обнулить последние i бит числа А

## Вычислить 2n

## Вычислить 2n+ 2m

## Установить i-ый бит числа A равным 1

## Инвертировать i-ый бит числа A

## Установить i-ый бит числа A равным 0

## Обнулить все кроме последних i битов числа A

## Определить значение i-ого бита числа A

## Вывести значение байта побитно