**ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ.**

Минимально адресуемая единица данных в микропроцессоре – **байт**. Логические команды позволяют манипулировать отдельными битами. Это единственные команды в системе команд микропроцессора, которые позволяют работать на битовом уровне. Этим, в частности, объясняется их важность. Булевы (логические) команды основаны на операциях булевой алгебры. Эти операции разрешают модификацию отдельных бит в двоичных числах, как показано в таблице:

|  |  |
| --- | --- |
| **Операция** | **Описание** |
| **AND** | Результат равен 1, только если оба бита равны 1. |
| **OR** | Результат равен 1, когда хотя бы один бит равен 1. |
| **XOR** | Результат равен 1, только когда оба бита разные (исключающее ИЛИ) |
| **NOT** | Результатом является противоположное значение (1 становится 0, а 0 становится 1) |
| **NEG** | Получает двоичное дополнение числа. |
| **TEST** | Выполняется команда AND без записи результат, устанавливаются только флаги. |
| **CMP** | Сравниваются два операнда, устанавливаются соответствующие флаги. |

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 источник обычно играет роль ***маски***. С помощью установленных в 1 битов этой маски и определяются нужные для конкретной операции биты операнда-приемника. Покажем, какие логические команды могут применяться для этой цели.

1. Для **установки определенных разрядов** (бит) применяется команда

**OR – Logical inclusive OR – логическое включающее ИЛИ.**

**OR *приемник, маска***

**Действие: *приемник = приемник OR маска.***

Команда выполняет операцию логического ИЛИ над соответствующими парами бит операндов ***приемник*** и ***маска.***

**Флаги: CF=0, OF=0, PF, AF не определен, ZF, SF.**

|  |  |  |  |
| --- | --- | --- | --- |
| **16-ричный код (1 байт)** | **MOD Reg/OPC Reg/Mem (2-ой байт)** | **смещение**  **disp\_Lo, disp\_Hi** | **формат операндов:**  **приемник, источник** |
| 08 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8/Mem8, Reg8 |
| 09 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16/Mem16, Reg16 |
| 0A | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8, Reg8/Mem8 |
| 0B | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16, Reg16/Mem16 |
| 0C | Data8 (непосредств.операнд) | отсутствует | AL, Immed8 |
| 0D | Data16 (непосред.операнд) | отсутствует | AX, Immed16 |
| 80 | MOD 001 Reg/Mem | Disp\_Lo, Disp\_Hi  Data Lo | Reg8/Mem8, Immed8 |
| 81 | MOD 001 Reg/Mem | Disp\_Lo, Disp\_Hi  Data Lo, Data Hi | Reg16/Mem16, Imm16 |
| 83 | MOD 001 Reg/Mem | Disp\_Lo, Disp\_Hi  Data SX | Reg16/Mem16, Imm8 |

**001** - расширение кода операции при работе с непосредственным операндом, размещается в поле Reg/Opc в байте адресации.

**Примеры команд:**

1. Мнемоническая запись: OR [BP+DI], AH ; формат операндов Mem8, Reg8

Машинный код команды: 00001000 00100011 ; команда занимает 2 байта

16-ричный код команды: 0823h.

2. Мнемоническая запись: OR DH, DH ; формат операндов Reg8, Reg8

Машинный код команды: 0000100011110110 ; команда занимает 2 байта

16-ричный код команды: 08F6h.

3. Мнемоническая запись: OR BX, [SI+12] ;формат операндов Reg16, Mem16

Машинный код команды: 000010110101110000001100 ;команда занимает 3 байта

16-ричный код команды: 0B5C0Ch.

4. Мнемоническая запись: OR AL, -1 ; формат операндов Reg8, imm8

Машинный код команды: 0000110011111111 ; команда занимает 2 байта

16-ричный код команды: 0CFFh.

2. С помощью команды **AND можно очищать отдельные биты** в операнде-приемнике, при этом защищая (***маскируя***) оставшиеся биты.

**AND – logical AND – логическое И.**

**Действие: *приемник = приемник AND маска.***

Воздействует на флаги **OF=0, SF, ZF, AF** не определен**, PF, CF=0.**

|  |  |  |  |
| --- | --- | --- | --- |
| **16-ричный код (1 байт)** | **MOD Reg/OPC Reg/Mem (2-ой байт)** | **смещение**  **disp\_Lo, disp\_Hi** | **формат операндов:**  **приемник, источник** |
| 20 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8/Mem8, Reg8 |
| 21 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16/Mem16, Reg16 |
| 22 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8, Reg8/Mem8 |
| 23 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16, Reg16/Mem16 |
| 24 | Data8 (непосредств.операнд) | отсутствует | AL, Immed8 |
| 25 | Data16 (непосред.операнд) | отсутствует | AX, Immed16 |
| 80 | MOD 100 Reg/Mem | Disp\_Lo, Disp\_Hi  Data Lo | Reg8/Mem8, Immed8 |
| 81 | MOD 100 Reg/Mem | Disp\_Lo, Disp\_Hi  Data Lo, Data Hi | Reg16/Mem16, Imm16 |
| 83 | MOD 100 Reg/Mem | Disp\_Lo, Disp\_Hi  Data SX | Reg16/Mem16, Imm8 |

**100** - расширение кода операции при работе с непосредственным операндом, размещается в поле Reg/Opc в байте адресации.

**Примеры команд:**

1. Мнемоническая запись: AND [BX+SI], CH ; формат операндов Mem8, Reg8

Машинный код команды: 00100000 00101000 ; команда занимает 2 байта

16-ричный код команды: 2028h.

2. Мнемоническая запись: AND BL, DL ; формат операндов Reg8, Reg8

Машинный код команды: 00100000 11010011 ; команда занимает 2 байта

16-ричный код команды: 20D3h.

3. Мнемоническая запись: AND DX, [BX+12] ;формат операндов Reg16, Mem16

Машинный код команды: 00100011 01010111 00001100 ;команда занимает 3 байта

16-ричный код команды: 23570Ch.

4. Мнемоническая запись: AND CL, [DI] ; формат операндов Reg8, Mem8

Машинный код команды: 00100010 00001101 ; команда занимает 2 байта

16-ричный код команды: 220Dh.

3. Команда **XOR приемник, источник**  применяется :

* для выяснения того, какие биты в приемнике и источнике различаются;
* для инвертирования состояния заданных бит в приемнике.

Интересующие нас биты ***маски*** (источник) при выполнении команды должны быть единичными, остальные – нулевыми. Например:

**xor ax, 10b ; инвертировать 1-ый бит в регистре АХ**

**jz mes ; переход, если 1-ый бит в АХ был единичным.**

**XOR – logical eXclusive OR – логическое исключающее ИЛИ.**

**XOR *приемник,источник***

**Действие: *приемник = приемник XOR источник.***

Команда выполняет операцию логического исключающего ИЛИ над соответствующими парами бит операндов ***приемник*** и ***источник.***

**Флаги: CF=0, OF=0, PF, AF не определен, ZF, SF.**

|  |  |  |  |
| --- | --- | --- | --- |
| **16-ричный код (1 байт)** | **MOD Reg/OPC Reg/Mem (2-ой байт)** | **смещение**  **disp\_Lo, disp\_Hi** | **формат операндов:**  **приемник, источник** |
| 30 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8/Mem8, Reg8 |
| 31 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16/Mem16, Reg16 |
| 32 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8, Reg8/Mem8 |
| 33 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16, Reg16/Mem16 |
| 34 | Data8 (непосредств.операнд) | отсутствует | AL, Immed8 |
| 35 | Data16 (непосред.операнд) | отсутствует | AX, Immed16 |
| 80 | MOD 110 Reg/Mem | Disp\_Lo, Disp\_Hi  Data Lo | Reg8/Mem8, Immed8 |
| 81 | MOD 110 Reg/Mem | Disp\_Lo, Disp\_Hi  Data Lo, Data Hi | Reg16/Mem16, Imm16 |
| 83 | MOD 110 Reg/Mem | Disp\_Lo, Disp\_Hi  Data SX | Reg16/Mem16, Imm8 |

**110** - расширение кода операции при работе с непосредственным операндом, размещается в поле Reg/Opc в байте адресации.

**Примеры команд:**

1. Мнемоническая запись: XOR [DI], CH ; формат операндов Mem8, Reg8

Машинный код команды: 00110000 00101101 ; команда занимает 2 байта

16-ричный код команды: 302Dh.

2. Мнемоническая запись: XOR AX, DI ; формат операндов Reg16, Reg16

Машинный код команды: 0011001111000111 ; команда занимает 2 байта

16-ричный код команды: 33C7h.

3. Мнемоническая запись: XOR BX, [SI+27h] ;формат операндов Reg16, Mem16

Машинный код команды: 001100110101110000100111 ;команда занимает 3 байта

16-ричный код команды: 335C27h.

4. Мнемоническая запись: XOR SI, 7FFFh ; формат операндов Reg16, imm16

Машинный код команды: 10000001111101101111111101111111 ; команда занимает 4 байта

16-ричный код команды: 81F6FF7Fh.

4. Для **проверки состояния** заданных бит применяется команда

**TEST приемник, источник**

Проверяемые биты ***приемника*** в ***маске (операнд-источник)*** должны иметь единичное значение. Алгоритм работы команды **TEST** подобен алгоритму работы команды **AND**, но он не меняет значение ***операнда-приемника***. Результатом работы является установка значения флага нуля **ZF**:

* **если ZF = 0**, то в результате логического умножения получился ненулевой результат, т.е. хотя бы один единичный бит ***маски*** совпал с соответствующим единичным битом ***приемника***;
* **если ZF = 1**, то в результате логического умножения получился нулевой результат, т.е. ни один единичный бит ***маски*** не совпал с соответствующим единичным битом ***приемника***.

Например:

**test ax, 0001h**

**jnz l1 ; переход если 0-ой бит равен 1 (переход, если флаг ZF<>0)**

1. Команда **NOT операнд**

делает **инверсию всех битов исходного операнда**, изменяя нули на единицы и наоборот. Результат называется ***дополнением до единицы***. Следующие операнды разрешены:

**NOT (NOT operand) – инвертирование всех битов операнда *приемник*.**

**NOT регистр**

**NOT память**

**Флаги: OF, SF, ZF, AF, PF, CF**

|  |  |  |  |
| --- | --- | --- | --- |
| **16-ричный код (1 байт)** | **MOD Reg/OPC Reg/Mem (2-ой байт)** | **смещение**  **disp\_Lo, disp\_Hi** | **формат операндов:**  **приемник, источник** |
| F6 | MOD 010 Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8/Mem8 |
| F7 | MOD 010 Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16/Mem16 |

**010** - расширение кода операции при работе с непосредственным операндом, размещается в поле Reg/Opc в байте адресации.

**Например,** MOV AX, 0FFFFh

NOT AX ; AX=0000h

MOV DI, 5551h

NOT DI ; DI=AAAEh

Например, дополнение до 1 числа F0h будет равняться 0Fh

**mov al, 11110000b**

**not al ; AL = 00001111b**

1. Команда **NEG операнд**

**инвертирует знак числа**, **преобразовывая число в его дополнение до двух**. Следующие операнды разрешены:

**NEG регистр**

**NEG память**

Подсчет дополнения до двух от числа может быть выполнен путем инверсии всех его битов и суммированием с единицей. После выполнения команды NEG проверьте флаг переполнения OF для определения правильности результата. Например, если поместить в регистр AL отрицательное число -128d, то в результате операции получим -128d (неправильный результат) и OF = 1:

* **mov al, -128 ; AL = 1000 0000b**
* **neg al ; AL = 1000 0000b, OF=1**

С другой стороны, если инвертировать +127, то результат правильный и OF=0:

* **mov al, +127 ; AL = 0111 1111b**
* **neg al ; AL = 1000 0001b, OF=0**

Операторы AND, OR, NOT и XOR выполняют поразрядные операции целых чисел во время трансляции. Операнд может быть целой константой или символом. Например, допустимы следующие выражения:

**X = 1101101b**

**Y = X AND 00001111b**

**Z = X OR 11110000b**

**Q = NOT Z**

**W = Y XOR 0FFFFh**





**6.** В следующем примере значение 0000 1111 является ***битовой маской***:

**mov al, 00111011b**

**and al, 00001111b ; AL = 00001011b**

***ПРИМЕР задания***

**1)** *Определить, является ли целое число степенью 2.*

Можно показать, что если *((x — 1)* ***AND*** *x)=0,* то *х* является степенью 2. Двоичные представления числа *х*, являющегося степенью 2, и соответствующего *х-1* приведены на рис.

|  |  |
| --- | --- |
| Двоичное x | Десятичное x |
| 10 | 2 |
| 100 | 4 |
| 1000 | 8 |
| 10000 | 16 |
| … | … |

|  |  |
| --- | --- |
| Двоичное x-1 | Десятичное x-1 |
| 1 | 1 |
| 11 | 3 |
| 111 | 7 |
| 1111 | 15 |
| … | … |

**Рис.** Двоичные и десятичные представления чисел

Легко видеть, что для таких х операция AND для х и х-1 дает в результате 0:

Теперь напишем фрагмент программы, который определяет, является ли число x степенью 2.

|  |  |
| --- | --- |
| AND | 00010000 |
| 00001111 |
|  | 00000000 |

.........

x dd ?

........

mov eax,x

dec eax

mov ebx,x

and ebx,eax

jz Power\_of\_Two

..............

Power\_of\_Two:

...............

**2)** *Выровнять адрес (число) на границу, кратную 2m, с округлением к младшим адресам.*

Предлагается следующее преобразование:

x  x AND NOT (2m -1)

Действительно, число 2m –1 в двоичном представлении — это m единиц (см. предыдущее задание).

Значит, NOT (2m –1) — это m нулей в младших разрядах, а в остальных (старших) разрядах — единицы.

Далее, если мы производим операцию AND для х с числом, у которого в младших разрядах m нулей, а в старших — 1, то это значит, что мы обнуляем младшие m разрядов. А это и есть выравнивание на границу, кратную 2^m.

Например,

x=1010111011110011 — двоичное число.

Пусть необходимо выровнять это число на границу, кратную 24, то есть до 3его двоичного разряда включительно.

m=4

2m-1 = 24-1 = 0000000000001111

NOT (24-1) = 1111111111110000

Теперь:

|  |  |
| --- | --- |
| AND | 1010111011110011 |
| 1111111111110000 |
|  | 1010111011110000 |

Напишем такой фрагмент программы для выравнивания адреса, записанного в переменной х длиной в слово, на границу параграфа с округлением к младшим адресам. Вспомним, что длина параграфа равна 16, то есть 24.

.........

x dw ?

........

mov ax,x

and ax,not(16-1)

..................

Получаем в ax число, округленное на границу параграфа с округлением к младшим адресам.

Обратите внимание на команду AND, а в ее операнде — операцию NOT.

**3)** *Выравнивание адреса на границу, кратную 2m, с округлением к старшим адресам.*

Предлагается следующее преобразование:

x  (x +(2m-1) AND NOT (2m-1))

Это аналогично предыдущему, только предварительно к х прибавляется число с единицами в младших m разрядах, чтобы увеличить m+1-ый разряд на 1.

Соответствующий фрагмент программы для выравнивания на границу параграфа с округлением к старшим адресам выглядит следующим образом:

........

x dw ?

........

mov ax,x

add ax,(16-1)

and ax,not(16-1)

................

**Команды сдвига**

При сдвиге биты операнда передвигаются влево или вправо в зависимости от команды.

Существуют три типа команд сдвига:

* Логический сдвиг — все биты операнда равноправны, выдвигаемые за пределы операнда биты пропадают, а последний заносится во флаг CF.
* Арифметический сдвиг — аналогичен логическому, но знаковый бит имеет особый статус.
* Циклический сдвиг — все биты равноправны, выдвигаемые из операнда биты не теряются, а вдвигаются с другой стороны.

1) **SHL** операнд, значение — логический сдвиг операнда влево (рис.1.19)

**SHL – SHift logical Left – логический сдвиг операнда влево.**

**SHL операнд, количество\_сдвигов**

Флаги: SF, ZF, PF, CF. Флаг AF не определен. Флаг OF изменяется, если сдвиг осуществляется только на 1 разряд, иначе не определен. OF=1, если значение флага CF после сдвига и значение старшего бита результата различны, если совпадают – 0 (т.е. XOR от CF и старшего разряда результата).

**Действие:**

Сдвиг влево всех битов операнда на указанное количество разрядов. При этом выдвигаемый слева бит становится значением флага CF. Если команда записана в формате:

**SHL *операнд, 1 -*** осуществляется сдвиг влево на 1 бит. В младший бит операнда загружается 0;

**SHL *операнд, CL -*** осуществляется сдвиг влево на число битов, указанное в регистре-счетчике CL. В процессе последовательных сдвигов старшие биты операнда, пройдя через флаг CF, теряются, а младшие заполняются 0.

Команда полностью эквивалента команде SAL, поэтому для нее действует та же таблица с кодами операций.

**Например,**  1) MOV AL, 0Ch

SHL AL, 1 ; AL=18h, CF=0

2) **SHR** операнд, значение — логический сдвиг вправо (рис.1.19)

**SHR – SHift logical operand Right – логический сдвиг операнда вправо.**

**SHR операнд, количество\_сдвигов**

Флаги: SF, ZF, PF, CF. Флаг AF не определен. Флаг OF устанавливается равным старшему значащему биту первоначального операнда. Флаг CF содержит значение последнего вдвинутого в него бита.

**Действие:**

Сдвиг вправо всех битов операнда. Младший бит операнда поступает в флаг CF. Освобождающиеся старшие разряды заполняются нулями.

Коды команды см. таблицу команды SAL. Но расширение кода операции при работе с непосредственным операндом, размещающееся в поле Reg/Opc в байте адресации, равно **101.**

**Например,**  1) MOV AL, 7h

SHR AL, 1 ; AL=3, CF=1

2) MOV DX, 9513h

MOV CL, 8

SHR DX, CL ; BX=0095h, CF=0

0

CF операнд

0

операнд CF

**Рис. 1.19** Логический сдвиг влево и вправо

Операнд определяет, *что* сдвигать (объект может быть 8,16 или 32-разрядный). Значение устанавливает, *на* сколько позиций сдвигать. Значение может быть записано непосредственно в команде, но тогда оно может быть только равно 1, либо значение может быть задано в регистре CL. Для МП i286 и выше непосредственно в командах сдвига может быть записана константа, большая 1.

При сдвиге выдвигаемые биты последовательно перемещаются во флаг CF. Если сдвиг более чем на 1 разряд, то в CF остается только последний выдвигаемый бит. При сдвиге влево выдвигаются старшие биты числа, при сдвиге вправо — младшие. На освободившееся место записываются нули.

***Примеры***

1)mov al,01010100b

shl al,1 ; результат — 10101000b, в CF выдвинулся 0

; и справа в результате добавился 0

2)mov cl,3

mov al,01010100b

shr al,cl ; результат -00001010b, в CF выдвинулась 1

; и слева добавилось три 0

3).386

shr al,3; в регистре al все биты сдвигаются на 3

; разряда

Команды сдвига позволяют производить быстрое умножение (сдвиг влево) или деление (сдвиг вправо) на числа, являющиеся степенями 2. В первом примере произошло умножение на 2, во втором примере — деление на 8.

Чтобы это «почувствовать», представим себе, что наш компьютер не «двоичный», а «десятичный» (рис.1.20). В этом случае при умножении на 10 мы добавляем справа 0, что эквивалентно сдвигу числа на 1 позицию влево (5\*10=50):

0

00005

00050

***Рис. 1.20*** *Умножение на 10 с помощью сдвига*

При делении удаляется справа 0, то есть происходит сдвиг числа вправо (500/10=050).

Это верно, если интерпретировать операнды как числа без знака. Для того чтобы можно было корректно работать с командами сдвига для чисел со знаком используют команды арифметического сдвига.

3) **SAL** операнд, значение — арифметический сдвиг влево (аналогично SHL)

В командах сдвигов в качестве операнда можно указывать любой регистр (кроме сегментного) или ячейку памяти размером как в байт, так и в слово. Не допускается использовать в качестве операнда непосредственное значение.

**SAL – Shift Arithmetic operand Left – арифметический сдвиг операнда влево.**

**SAL операнд, количество\_сдвигов**

Флаги: SF, ZF, PF, CF. Флаг AF не определен. Флаг OF изменяется, если сдвиг осуществляется только на 1 разряд, иначе не определен. OF=1, если значение флага CF после сдвига и значение старшего бита результата различны, если совпадают – 0 (т.е. XOR от CF и старшего разряда результата).

**Действие:**

Сдвиг влево всех битов операнда. Каждый сдвиг влево эквивалентен умножению знакового числа на 2, поэтому команду удобно использовать для возведения операнда в степень 2. Старший бит операнда поступает в флаг CF. Если команда записана в формате:

**SAL *операнд, 1 -*** осуществляется сдвиг влево на 1 бит. В младший бит операнда загружается 0;

**SAL *операнд, CL -*** осуществляется сдвиг влево на число битов, указанное в регистре-счетчике CL. В процессе последовательных сдвигов старшие биты операнда, пройдя через флаг CF, теряются, а младшие заполняются 0;

|  |  |  |  |
| --- | --- | --- | --- |
| **16-ричный код (1 байт)** | **MOD Reg/OPC Reg/Mem (2-ой байт)** | **смещение**  **disp\_Lo, disp\_Hi** | **формат операндов:**  **приемник, источник** |
| D0 | MOD 100 Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8/Mem8, 1 |
| D1 | MOD 100 Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16/Mem16, 1 |
| D2 | MOD 100 Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8/Mem8, CL |
| D3 | MOD 100 Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16/Mem16, CL |
| C0 | MOD 100 Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8/Mem8, imm8 |
| C1 | MOD 100 Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16/Mem16, imm8 |

**100** - расширение кода операции при работе с непосредственным операндом, размещается в поле Reg/Opc в байте адресации.

**Например,**  1) MOV AL, 7h

SAL AL, 1 ; AL=0Eh=7\*2, CF=0

1. MOV AX, -1 ; AX=FFFFh

MOV CL, 4

SAL AX, CL ; AX=FFF0h= -1\*16= -16, CF=1

4) **SAR** операнд, значение — арифметический сдвиг вправо (рис.1.21)

**SAR – Shift Arithmetic operand Right – арифметический сдвиг операнда вправо.**

**SAR операнд, количество\_сдвигов**

Флаги: SF, ZF, PF, CF. Флаг AF не определен. Флаг OF=0, если сдвиг осуществляется только на 1 разряд, иначе не определен.

**Действие:**

Сдвиг вправо всех битов операнда. Младший бит операнда поступает в флаг CF. Освобождающиеся старшие разряды заполняются значением старшего (знакового) разряда. Каждый сдвиг вправо эквивалентен делению знакового числа на 2, поэтому команду удобно использовать для деления операнда на целые степени 2.

Коды команды см. таблицу команды SAL. Но расширение кода операции при работе с непосредственным операндом, размещающееся в поле Reg/Opc в байте адресации, равно **111.**

**Например,**  1) MOV AL, 7h

SAR AL, 1 ; AL=3=7/2, CF=1. Остаток потерян.

1. MOV BX, -8 ; BX=FFF8h

MOV CL, 2

SAR BX, CL ; BX=FFFEh= -2= -8/4, CF=0

знак

операнд CF

***Рис. 1.21*** *Арифметический сдвиг вправо*

При работе команды арифметического сдвига вправо значение знакового бита не меняется, знаковый бит заполняет освободившиеся места.

***Пример***

mov al,11010100b

mov cl,3

sar al,cl ; результат -11111010

mov al,11010100b

shr al,cl ; результат — 00011010 — знак числа потерялся

Таким образом, можно производить умножение и деление с помощью сдвига, если один из сомножителей или делитель представляет собой степень 2. Причем, **сдвиг производится процессором намного быстрее, чем умножение и деление:**

MUL (8-битный регистр) — 70— 77 тактов

(16-битный) —118—133 тактов

IMUL (8-битный регистр) — 80— 98 тактов

(16-битный) —128—154 тактов

IDIV (8-битный регистр) —101—112 тактов

(16-битный) —165—184 тактов

сдвиг регистра на 1 позицию — 2 такта

Даже если мы работаем с числами, не являющимися степенями 2, оказывается, если мы хотим добиться максимальной скорости, можно представить множитель или делитель как сумму степеней 2.

***ПРИМЕР задания***

*Написать фрагменты программы для умножения X\*12 двумя способами: с помощью команды умножения и с помощью сдвига. Оценить время выполнения каждого фрагмента.*

X\*12 = X \* (23 + 22) = X \* 23 + X \* 22

Чтобы умножить число X на 12, нужно произвести сдвиг X влево на 3 бита, и результат где-то запомнить, затем произвести сдвиг X влево на 1 бит. Потом оба результата сложить.

x db 30 x db 30

.......... ..........

mov al,x mov al,x

cbw

mov cl,2 cbw

sal ax,cl ; ax=ax\*4 mov bx,10; 2 такта

mov bx,ax imul bx; 128 тактов

sal ax,1 ; ax=ax\*2

add bx,ax **всего** 134 такта

cbw, sal, mov — 2 такта

add — 3 такта

**всего** 15 тактов

Таким образом, хоть первая программа и длинней, она будет выполняться намного быстрее (за 15 тактов), а вторая — за 134 такта. Отметим, что здесь не учитывается время для выборки переменной из сегмента данных.

5) **ROL** операнд, значение — сдвиг влево циклический (ROll Left) (рис.1.22)

**ROL – Rotate operand Left – циклический сдвиг операнда влево.**

**ROL операнд, количество\_сдвигов**

Флаги: SF, ZF, PF, CF. Флаг AF не определен.

**Машинные коды см. таблицу операции SAL.** Расширение кода операции при работе с непосредственным операндом, размещающееся в поле Reg/Opc в байте адресации, равно **000.**

**Например,**  1) MOV DL, 80h

ROL DL, 1 ; DL=01h, CF=1

1. MOV DX, 2000h

MOV CL, 4

ROL DX, CL ; DX=0002h, CF=0

6) **ROR** операнд, значение — сдвиг вправо циклический (ROll Right) (рис.1.22)

**ROR – Rotate operand Right – циклический сдвиг операнда вправо.**

**ROR операнд, количество\_сдвигов**

Флаги: SF, ZF, PF, CF. Флаг AF не определен.

Коды команды см. таблицу команды SAL. Но расширение кода операции при работе с непосредственным операндом, размещающееся в поле Reg/Opc в байте адресации, равно **001.**

**Например,**  1) MOV DL, 81h

ROR DL, 1 ; DL=C0h, CF=1

1. MOV BX, 000Eh

MOV CL, 4

ROR BX, CL ; BX=E000, CF=1

CF

CF

***Рис. 1.22*** *Циклические сдвиги влево и вправо*

При циклическом сдвиге биты не пропадают, а вдвигаются с другой стороны.

7) **RCL** операнд, значение — сдвиг влево циклический через перенос (Roll Cf Left)

**RCL – Rotate operand through Carry flag Left – циклический сдвиг операнда влево через флаг переноса.**

**RCL операнд, количество\_сдвигов**

Флаги: SF, ZF, PF, CF. Флаг AF не определен.

**Машинные коды см. таблицу операции SAL.** Расширение кода операции при работе с непосредственным операндом, размещающееся в поле Reg/Opc в байте адресации, равно **010.**

**Например,**  1) MOV DL, 7

RСL DL, 1 ; DL=0Fh, CF=0 (предыдущее значение 1)

1. MOV DH, 3

MOV CL, 4

RСL DH, CL ; DH=30h, CF=0 (предыдущее значение 0)

8) **RCR** операнд, значение — сдвиг вправо циклический через перенос (Roll Cf Right)

**RСR – Rotate operand through Carry flag Right – циклический сдвиг операнда вправо через флаг переноса.**

**RСR операнд, количество\_сдвигов**

Флаги: SF, ZF, PF, CF. Флаг AF не определен.

Коды команды см. таблицу команды SAL. Но расширение кода операции при работе с непосредственным операндом, размещающееся в поле Reg/Opc в байте адресации, равно **011.**

***Пример 1*** 1) MOV DL, 8h

RСR DL, 1 ; DL=84h, CF=0 (предварительно установлен)

1. MOV BH, 80h

MOV CL, 5

RCR BH, CL ; BH=4, CF=0 (предварительно сброшен).

CF

CF

***Рис. 1.23*** *Циклические сдвиги через перенос влево и вправо*

Команды RCL, RCR включают CF в кольцо.

***Пример 2***

*mov al,11010101b*

*rol al,1; результат 10101011, в CF — 1, и она же — в 0 бит*

*mov al,11010101b; CF=0*

*rcr al,1; результат 01101010, из CF в 7 бит — 0, в CF — 1*

Команды циклического сдвига могут использоваться, когда необходимо, например, менять местами фрагменты битов в операнде.

***Пример 3***

EAX

#### FFFE 000C

Младшая часть регистра EAX имеет название — AX, а старшая — нет. Команда ROL позволяет получить доступ к старшей части 32х-разрядного регистра:

*ROL EAX,16 ; младшая и старшая часть регистра поменяются местами*

Теперь:

000C FFFE

EAX

AX

и бывшая старшая часть доступна в AX.

***Пример 4***

*Поделить число, находящееся в паре регистров DX, AX, на 4.*

Чтобы это осуществить, нужно синхронно сдвигать оба регистра вправо так, чтобы биты из DX попадали при сдвиге в AX. Один из способов состоит в следующем:

1) сдвигаем DX на 1 разряд вправо (при этом бит из DX попадает во флаг CF);

2) осуществляем циклический сдвиг вправо через перенос регистра AX, при этом бит из CF (он туда попал из DX) попадает в старший разряд регистра AX.

Пункты 1) и 2) повторяем 2 раза, так как 4=22.

*..........*

*shr dx,1*

*rcr ax,1*

*shr dx,1*

*rcr ax,1*

*..........*

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Что такое маска?
2. Приведите примеры ситуаций, в которых необходимо использование логических команд.
3. Команда OR, как создать маску, принцип использования?
4. Команда AND, как создать маску, принцип использования?
5. Команда XOR, как создать маску, принцип использования?
6. В чем разница между командами NOT и NEG?
7. В каких ситуациях используют команду сравнения? Принцип ее работы.
8. В чем отличие между командами TEST и AND?
9. Какие флаги используются при сравнении чисел без знака?
10. Какие флаги используются при сравнении чисел со знаком?
11. Как использовать команду OR для проверки знака числа или равенства 0 числа?
12. Назовите и объясните принципы работы команд сдвигов (арифметических, логических, циклических).

**ПРАКТИЧЕСКИЕ ЗАДАНИЯ**

***Задание 1***

Записать машинные коды команд, их 16-ричное представление, выполнить действия и установить флаги.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *№ вар*. | *Пример 1* | *Пример 2* | *Пример 3* | *Пример 4* |
| **1, 2, 27** | SHL AX, 1 | SAL DI, CL | RCR CX,1 **(CF=1)** | AND 8[SI][DI], BH **(4)** |
| **3, 4, 26** | AND CX, DI | OR AX, 0F0Dh | ROR DI, 1 | RCR DI, 1 **(CF=0)** |
| **5, 6, 24** | ROL CL,1 | SHL BH, 1 | OR byte ptr [DI], 5 **(2)** | AND AL, 4 |
| **7, 8, 23** | NOT AH | ROL CX, 1 | ROL AX, CL | SHL DL, CL |
| **9, 10, 22** | SAL DX, 1 | AND DX, 0DBBh | OR [DI], AH **(3)** | ROR BX, 1 |
| **11,12,21** | SAR AL, 1 | RCL SI, 1 **(CF=0)** | SAR SI, CL | OR CX, 0AAh |
| **13,14,20** | OR DI, [SI+24] **(1)** | SHR SI, 1 | ROR DX, CL | NOT word ptr [BP] **(5)** |
| **15,16,19** | AND AX, SI | SHL BX, 1 | OR [SI], AL **(3)** | SAL DH, CL |
| **17,18,25** | OR byte ptr[BX],98 | AND CH,05Ah | SAL BL,CL | ROR AX,1 |

*Содержимое регистров:*

(AX) → 2A69h

(BX) → 3FF7h

(CX) → 7D06h

(DX) → 5B7Ch

(SI) → 0F95h

(DI) → 2E75h

(BP) → 00FFh

*Содержимое ячеек памяти:*

(1) → AA88h

(2) → B1h

(3) → 7Fh

(4) → 80h

(5) → C69Eh

***Пример выполнения:***

Записать машинные коды команд, их 16-ричное представление, выполнить действия и установить флаги.

AND [SI], CH ; ( [SI] ) → 59h, (CH) → 3Dh

Машинный код команды: 00100000 00101100 ; команда занимает 2 байта

16-ричный код команды: 202Ch.

Выполним логическую команду AND над операндами

мы получили следующий результат: **19h.**

**Установим флаги, на которые влияет операция AND:**

1. CF=0 (всегда);
2. PF = 0, т.к. кол-во 1 в байте результата = 3, а это число нечетное;
3. AF не определен;
4. ZF = 0, т.к. результат операции не равен нулю;
5. SF = 0, т.к. знаковый бит содержит 0;
6. OF = 0 (всегда).

***Задание 2 (по вариантам)*.**

1. В регистре DL установить 6-й, 3-й и 1-й биты и сбросить 0 бит.
2. Инвертировать 4-й и 3-й биты регистра BH, а знаковый бит установить в 1.
3. Умножить операнд в DL на 10.
4. Проверить делимость числа на 2
5. Двойное слово в DX:AX умножить на 4.
6. Сосчитать количество единичных битов в AX. Результат поместить в BX.
7. Разделить содержимое регистра CX на 32.
8. Поместить в регистр CL старший байт регистра DI.
9. Сосчитать количество единичных битов в AX.
10. Расположить содержимое AX в регистре BX в обратном порядке.
11. Инвертировать 3-й и 0-й биты регистра CH и сбросить 5 и 7 биты.
12. Проверить делимость числа на 4.
13. Двойное слово в DX:AX разделить на 8.
14. В регистре AL установить 1-й, 5-й и 6-й биты, а 4 бит инвертировать.
15. Получить дополнительный код числа в регистре DI, не используя команды NOT или NEG.
16. Заменить три старших бита регистра SI на три младших бита регистра AX.
17. Умножить операнд в DH на 9.
18. Инвертировать четные биты регистра DX, а нечетные обнулить.
19. Разделить содержимое AX на 64.
20. Проверить делимость содержимого регистра SI на 16.
21. Умножить содержимое регистра BX на 14.
22. Инвертировать нечетные биты регистра AX, а четные установить в 1.
23. Сосчитать количество нулевых битов в CX. Результат поместить в DX.
24. Поменять местами содержимое регистров AL и AH.
25. Поместить в регистр BL младший байт регистра AX
26. Умножить содержимое регистра SI на 11.
27. Проверить делимость числа на 8.
28. В регистре CX в четные биты записать 1, а в нечетные 0.

*Содержимое регистров:*

(AX) → 2A69h; (BX) → 3FF7h; (CX) → 7D06h; (DX) → 5B7Ch; (SI) → 0F95h; (DI) → 2E75h

**При оформлении отчета в распечатке листинга рядом с логическими командами и командами сдвигов выписать состояние флагов после их выполнения.**