**ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ.**

### Двоичная арифметика

Все команды из этого раздела, кроме команд деления и умножения, изменяют флаги OF, SF, ZF, AF, CF, PF в соответствии с назначением каждого из этих флагов (см. главу 2.1.4).

|  |  |
| --- | --- |
| Команда: | **ADD** приемник, источник |
| Назначение: | Сложение |
| Процессор: | 8086 |

Команда выполняет арифметическое сложение приемника и источника, помещает сумму в приемник, не изменяя содержимое источника. Приемник может быть регистром или переменной, источник может быть 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), можно использовать ее и для тех, и для других.

|  |  |
| --- | --- |
| Команда: | **ADC** приемник, источник |
| Назначение: | Сложение с переносом |
| Процессор: | 8086 |

Эта команда во всем аналогична ADD, кроме того, что она выполняет арифметическое сложение приемника, источника и флага СF. Пара команд ADD/ADC используется для сложения чисел повышенной точности. Сложим, например, два 64-битных целых числа: пусть одно из них находится в паре регистров EDX:EAX (младшее двойное слово (биты 0 – 31) — в ЕАХ и старшее (биты 32 – 63) — в EDX), а другое — в паре регистров ЕВХ:ЕСХ:

*add eax,ecx*

*adc edx,ebx*

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

|  |  |
| --- | --- |
| Команда: | **SUB** приемник, источник |
| Назначение: | Вычитание |
| Процессор: | 8086 |

Вычитает источник из приемника и помещает разность в приемник. Приемник может быть регистром или переменной, источник может быть 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 ее как для тех, так и для других.

|  |  |
| --- | --- |
| Команда: | **SBB** приемник, источник |
| Назначение: | Вычитание с займом |
| Процессор: | 8086 |

Эта команда во всем аналогична SUB, кроме того, что она вычитает из приемника значение источника и дополнительно вычитает значение флага CF. Так, можно использовать эту команду для вычитания 64-битных чисел в EDX:EAX и ЕВХ:ЕСХ аналогично ADD/ADC:

*sub eax,ecx*

*sbb edx,ebx*

Если при вычитании младших двойных слов произошел заем, то он будет учтен при вычитании старших.

|  |  |
| --- | --- |
| Команда: | **IMUL** источник |
| Назначение: | Умножение чисел со знаком |
| Процессор: | 8086 |

Источник (регистр или переменная) умножается на AL, АХ или ЕАХ (в зависимости от размера операнда), и результат располагается в АХ, DX:AX или EDX:EAX соответственно.

Считается, что результат может занимать в два раза больше места, чем размер источника. В результате умножения могут произойти переполнение и потеря старших бит результата. Флаги OF и CF будут равны единице, если это произошло, и нулю, если результат умножения поместился целиком в приемник (во втором и третьем случаях) или в младшую половину приемника (в первом случае).

Значения флагов SF, ZF, AF и PF после команды IMUL не определены.

|  |  |
| --- | --- |
| Команда: | **MUL** источник |
| Назначение: | Умножение чисел без знака |
| Процессор: | 8086 |

Выполняет умножение содержимого источника (регистр или переменная) и регистра AL, АХ, ЕАХ (в зависимости от размера источника) и помещает результат в АХ, DX:AX, EDX:EAX соответственно. Если старшая половина результата (АН, DX, EDX) 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.

|  |  |
| --- | --- |
| Команда: | **IDIV** источник |
| Назначение: | Целочисленное деление со знаком |
| Процессор: | 8086 |

Выполняет целочисленное деление со знаком AL, АХ или ЕАХ (в зависимости от размера источника) на источник (регистр или переменная) и помещает результат в AL, АХ или ЕАХ, а остаток — в АН, DX или EDX соответственно. Результат всегда округляется в сторону нуля, знак остатка всегда совпадает со знаком делимого, абсолютное значение остатка всегда меньше абсолютного значения делителя. Значения флагов 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 0 — в реальном.

|  |  |
| --- | --- |
| Команда: | **DIV** источник |
| Назначение: | Целочисленное деление без знака |
| Процессор: | 8086 |

Выполняет целочисленное деление без знака AL, АХ или ЕАХ (в зависимости от размера источника) на источник (регистр или переменная) и помещает результат в AL, АХ или ЕАХ, а остаток — в АН, DX или EDX соответственно. Результат всегда округляется в сторону нуля, абсолютное значение остатка всегда меньше абсолютного значения делителя. Значения флагов 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 0 — в реальном.

|  |  |
| --- | --- |
| Команда: | **INC** приемник |
| Назначение: | Инкремент |
| Процессор: | 8086 |

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.

|  |  |
| --- | --- |
| Команда: | **DEC** приемник |
| Назначение: | Декремент |
| Процессор: | 8086 |

Уменьшает приемник (регистр или переменная) на 1. Единственное отличие этой команды от SUB приемник,1 состоит в том, что флаг CF не затрагивается. Остальные арифметические флаги (OF, SF, ZF, AF, PF) устанавливаются в соответствии с результатом вычитания.

|  |  |
| --- | --- |
| Команда: | **NEG** приемник |
| Назначение: | Изменение знака |
| Процессор: | 8086 |

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) устанавливаются в соответствии с результатом операции.

***Примечание:*** Красивый пример применения команды NEG — получение абсолютного значения числа, используя всего две команды — изменение знака и переход на первую команду еще раз, если знак отрицательный:

*label0: neg eax*

*js label0*

### Десятичная арифметика

Процессоры Intel поддерживают операции с двумя форматами десятичных чисел: ***неупакованное двоично-десятичное число*** — байт, принимающий значения от 00 до 09, и ***упакованное двоично-десятичное число*** — байт, принимающий значения от 00 до 99h. Все обычные арифметическиe 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 следующие команды.

|  |  |
| --- | --- |
| Команда: | **DAA** |
| Назначение: | BCD-коррекция после сложения |
| Процессор: | 8086 |

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

*inc al*

*daa*

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

***Примечание:*** DAA выполняет следующие действия:

* 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.
* Иначе AF = 0.
* Если теперь старшие четыре бита AL больше 9 или флаг CF = 1, то AL увеличивается на 60h и CF устанавливается в 1.
* Иначе CF = 0.

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

|  |  |
| --- | --- |
| Команда: | **DAS** |
| Назначение: | BCD-коррекция после вычитания |
| Процессор: | 8086 |

Если эта команда выполняется сразу после SUB (SBB или DEC) и в регистре AL находится разность двух упакованных двоично-десятичных чисел, то в результате 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 AL окажется 19h (а не 1Fh, как было бы после DEC).

***Примечание:*** DAS выполняет следующие действия:

* 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.
* Иначе AF = 0.
* Если теперь старшие четыре бита AL больше 9 или флаг CF = 1, то AL уменьшается на 60h и CF устанавливается в 1.
* 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 (т.к. CF=1 после CMP), а числа 0Ah – 0Fh — в 0А1h – 0A6h (т.к. CF=0 после CMP). Затем DAS вычитает 66h из первой группы чисел, переводя их в 30h – 39h, и 60h из второй группы чисел, переводя их в 41h – 46h.

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

|  |  |
| --- | --- |
| Команда: | **AAA** |
| Назначение: | ASCII-коррекция после сложения |
| Процессор: | 8086 |

Корректирует сумму двух неупакованных двоично-десятичных чисел в AL. Если коррекция приводит к десятичному переносу, АН увеличивается на 1. Эта команда имеет смысл сразу после команды сложения двух таких чисел. Например, если при сложении 05 и 06 в АХ окажется число 000Bh, то команда ААА скорректирует его в 0101h (неупакованное десятичное 11). Флаги CF и OF устанавливаются в 1, если произошел перенос из AL в АН, иначе они равны нулю. Значения флагов OF, SF, ZF и PF не определены.

|  |  |
| --- | --- |
| Команда: | **AAS** |
| Назначение: | ASCII-коррекция после вычитания |
| Процессор: | 8086 |

Корректирует разность двух неупакованных двоично-десятичных чисел в AL сразу после команды SUB или SBB. Если коррекция приводит к займу, АН уменьшается на 1. Флаги CF и OF устанавливаются в 1, если произошел заем из AL в АН, и в ноль — в противном случае. Значения флагов OF, SF, ZF и PF не определены.

|  |  |
| --- | --- |
| Команда: | **AAM** |
| Назначение: | ASCII-коррекция после умножения |
| Процессор: | 8086 |

Корректирует результат умножения неупакованных двоично-десятичных чисел, находящийся в АХ после выполнения команды MUL, преобразовывая полученный результат в пару неупакованных двоично-десятичных чисел (в АН и AL). Например:

*mov al,5*

*mov bl,5 ; умножить 5 на 5*

*mul bl ; результат в АХ - 0019h*

*aam ; теперь АХ содержит 0205h*

ААМ устанавливает флаги SF, ZF и PF в соответствии с результатом и оставляет OF, AF и CF неопределенными.

***Примечание:*** Код команды ААМ — D4h 0Ah, где 0Ah — основание системы счисления, по отношению к которой выполняется коррекция. Этот байт можно заменить на любое другое число (кроме нуля), и ААМ преобразует АХ к двум неупакованным цифрам любой системы счисления. Такая обобщенная форма ААМ работает на всех процессорах (начиная с 8086), но появляется в документации Intel только с процессоров Pentium. Фактически действие, которое выполняет ААМ, — целочисленное деление AL на 0Ah (или любое другое число в общем случае), частное помещается в AL, и остаток — в АН, так что эту команду часто используют для быстрого деления в высокооптимизированных алгоритмах.

|  |  |
| --- | --- |
| Команда: | **AAD** |
| Назначение: | ASCII-коррекция перед делением |
| Процессор: | 8086 |

Выполняет коррекцию неупакованного двоично-десятичного числа, находящегося в регистре АХ, так, чтобы последующее деление привело к корректному десятичному результату. Например, разделим десятичное 25 на 5:

*mov ax,0205h ; 25 в неупакованном формате*

*mov bl,5*

*aad ; теперь в АХ находится 19h*

*div bl ; АХ = 0005*

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

***Примечание:*** Так же как и команда ААМ, AAD используется с любой системой счисления: ее код — D5h 0Ah, и второй байт можно заменить на любое другое число. Действие AAD состоит в том, что содержимое регистра АН умножается на второй байт команды (0Ah по умолчанию) и складывается с AL, после чего АН обнуляется, так что AAD можно использовать для быстрого умножения на любое число.

**ПРАКТИЧЕСКИЕ ЗАДАНИЯ**

***Задание 1.***

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

|  |  |  |  |
| --- | --- | --- | --- |
| **№ вар.** | **Пример 1** | **Пример 2** | **Пример 3** |
| 1,2,27 | ADD AL, 17h  DAA | DEC BX | SUB [DI], 0DBBh **(3)** |
| 3,4,26 | ADD AL, 4  DAA | SUB CX, 0F0Dh | INC word ptr [DI] **(4)** |
| 5,6,25 | DEC CX | ADD DX, 0A4Fh | SUB AL, 39h  DAS |
| 7,8,24 | INC AX | ADC AH,105 **(CF=1)** | SUB AL, 81h  DAS |
| 9,10,23 | SUB [DI], AH **(1)** | ADD AL, 90h  DAA | DEC SI |
| 11,12,22 | SUB AL, 67h  DAS | DEC word ptr [BX] **(4)** | ADC DX, AX **(CF=1)** |
| 13,14,21 | ADD BX, [BP+3Fh] **(2)** | DEC AL  DAS | SBB DX, 0FD54h **(CF=1)** |
| 15,16,20 | SUB AL, BH  DAS | INC DI | ADD 34[DI], BL **(1)** |
| 17,18,19 | SUB BX, SI | ADC SI, AX **(CF=0)** | INC AL  DAA |

**Содержимое регистров: Содержимое ячеек памяти:**

**(AX) → 8899h (SI) → 00DDh (1) → 6Bh**

**(BX) → 5F5Eh (DI) → 753Ah (2) → E10Fh**

**(CX) → 3399h (BP) → 0016h (3)→ 5F7Fh**

**(DX) → F070h (4)→ 0000h**

***Задание 2.***

Разработать программу реализующую указанную формулу, исполнить программу с несколькими (три-четыре) наборами исходных данных, проверить правильность результатов.

1. Х= А - 5 (В - 2С) + 2
2. Х= - 4А + (В + С) / 4 + 2
3. Х= 7А - 2В - 100 + С
4. Х= - А / 2 + 4 (В + 1) + 3С
5. Х= 5 (А - В) - 2С + 5
6. Х= (А/ 2 + В) / 4 + С – 1
7. Х= - (С + 2А + 4В + В)
8. Х= 6С + (В - С + 1) / 2
9. Х= 2 - В (А + В) + С / 4
10. Х= 2В - 1 + 4 (А - 3С)
11. Х= (2А + В) / 4 - С / 2 + 168
12. Х= 6 (А - 2В + С / 4) + 10
13. Х= 5 (А - В ) + С mod 4
14. Х=- ( - (С + 2А) \* 4В + 38)
15. Х= А - 3 (А + В) + С mod 4
16. Х= 3(А - 2В) +50 – С / 2
17. Х= (3А + 2В) - С / 4 + 217
18. Х= 3(С - 2A) + (В - С + 1) / 2
19. Х= (2А + В) / 4 - С / 2 + 168
20. Х= 6 (А - 2В + С / 4) + 10
21. Х= 3 (А - 4В ) + С / 4
22. Х= - ( - (С + 2А) \* 5В - 27)
23. Х= А / 2 - 3 (А + В) + С \* 4
24. Х= 3(А - 2В) +50 – С /2
25. Х= 5А + 2В - B / 4 + 131

***Пример выполнения задания 2***

***Вычислить Х = 3А + ( В + 5 ) / 2 - С - 1, где А, В, С, Х- целые знаковые числа занимающие слово, написать программу реализующую данную формулу.***

*Распишем формулу по отдельным операциям:*

*АХ ← А ; значение А в регистре АХ*

*АХ ← 2 \*( АХ ) ; 2А в АХ*

*АХ ← ( АХ ) + А ; 3А в АХ*

*ВХ ← В ; В в ВХ*

*ВХ ← 5 + ( ВХ ) ; В+5 в ВХ*

*ВХ ← (ВХ ) / 2 ; ( В+5) / 2 в ВХ*

*АХ ← (BX ) + ( AX ) ; 3А+( В+5 ) / 2 в АХ*

*АХ ← ( АХ ) - С ; 3А+( В+5 ) / 2 - С в АХ*

*АХ ← ( АХ ) -1 ; 3А+( В+5 )/2 - С - 1 в АХ*

*Х ← ( АХ ) ; 3А+( В+5 )/2 - С - 1 в Х*

*Ниже приведена типичная структура простой программы на ассемблере*

***data segment***

*a dw 10*

*b dw 20*

*c dw 5*

*x dw ?*

***data ends***

***code segment***

*assume cs: code, ds: data*

*start: mov ax, data*

*mov dx, ax ; загрузить адрес*

*mov ax, a ; сегмента данных*

*sal ax, 1 ; умножение на 2 (сдвиг влево на 1 разряд)*

*add ax, a*

*mov bx, b*

*add bx, 5*

*sar bx, 1 ; деление на 2 (сдвиг вправо на 1 разряд)*

*add ax, bx*

*sub ax, c*

*dec ax*

*mov x, ax ; запись результата в память*

*quit:*

*mov ax, 4c00h ; код завершения 0*

*int 21 ; выход в dos*

***code ends***

*end start*

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Назовите арифметические команды, выполняемые процессором.
2. Назначение директив SEGMENT и ENDS.
3. Назначение директивы ASSUME.
4. Назначение директив DB, DW.
5. Назначение оператора DUP в директивах DB, DW.
6. Назначение директивы END.
7. Из каких полей состоит строка программы на ассемблере?
8. Какие поля обязательны, а какие можно опустить?
9. В чем различие между командами :

mov AX, BX ; mov AX, [ BX ] и mov [ AX ], BX ?

1. Какие ограничения накладываются на выполнение команды сложения?
2. В чем разница между командой mov A, 1 и директивой А dw 1 ?
3. Прокомментируйте влияние на флаги команд арифметического сложения и вычитания.
4. Прокомментируйте понятие «расширение кода операции».
5. Где могут размещаться операнды у одноадресных команд?
6. Объясните назначение двоично-десятичных операндов.
7. Каким образом могут храниться BCD-числа?
8. Зачем нужны операции коррекции кода при работе с двоично-десятичными числами?
9. Как работает команда коррекции DAA?
10. Приведите пример использования команды ADC.
11. Показать, что для чисел FAh и 01h операция вычитания, аналогично сложению, также выполнится правильно как в интерпретации для чисел со знаком, так и для чисел без знака.
12. Произвести вычитание двух 64х-разрядных чисел на 32х-разрядном процессоре.
13. Написать программу для умножения 2х чисел 25h и FFh как чисел без знака. Какие флаги изменятся после выполнения этой программы?
14. Какие команды содержат ошибку?

1) mov byte ptr es:[bx+2],20

2) sub 20,byte ptr es:[bx+2]

3) mov al,word ptr es:[bx]

4) mov dx,word ptr es:[cx]

5) add ax,word ptr cs:[bx]

15. Задан фрагмент программы:

N DB ?

D DB 3 DUP(?)

Рассматривая N как число без знака, записать в массив D цифры (как символы) из десятичной записи этого числа: в байт D — левую цифру, в D + 1 — среднюю цифру, в D + 2 — правую цифру.

ПРИЛОЖЕНИЕ А

**ТАБЛИЦЫ КОДИРОВОК АРИФМЕТИЧЕСКИХ КОМАНД**

**ADD – ADDition – сложение двух целочисленных двоичных операндов**.

|  |  |  |  |
| --- | --- | --- | --- |
| **16-ричный код (1 байт)** | **MOD Reg/OPC Reg/Mem (2-ой байт)** | **смещение**  **disp\_Lo, disp\_Hi** | **формат операндов:**  **приемник, источник** |
| 00 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8/Mem8, Reg8 |
| 01 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16/Mem16, Reg16 |
| 02 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8, Reg8/Mem8 |
| 03 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16, Reg16/Mem16 |
| 04 | Data8 (непосредств.операнд) | отсутствует | AL, Immed8 |
| 05 | Data16 (непосред.операнд) | отсутствует | AX, Immed16 |
| 80 | MOD 000 Reg/Mem | Disp\_Lo, Disp\_Hi  Data Lo | Reg8/Mem8, Immed8 |
| 81 | MOD 000 Reg/Mem | Disp\_Lo, Disp\_Hi  Data Lo, Data Hi | Reg16/Mem16, Imm16 |
| 83 | MOD 000 Reg/Mem | Disp\_Lo, Disp\_Hi  Data SX | Reg16/Mem16, Imm8 |

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

**ADC – Addition with Carry – сложение с учетом значения флага переноса CF.**

|  |  |  |  |
| --- | --- | --- | --- |
| **16-ричный код (1 байт)** | **MOD Reg/OPC Reg/Mem (2-ой байт)** | **смещение**  **disp\_Lo, disp\_Hi** | **формат операндов:**  **приемник, источник** |
| 10 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8/Mem8, Reg8 |
| 11 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16/Mem16, Reg16 |
| 12 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8, Reg8/Mem8 |
| 13 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16, Reg16/Mem16 |
| 14 | Data8 (непосредств.операнд) | отсутствует | AL, Immed8 |
| 15 | Data16 (непосред.операнд) | отсутствует | AX, Immed16 |
| 80 | MOD 010 Reg/Mem | Disp\_Lo, Disp\_Hi  Data Lo | Reg8/Mem8, Immed8 |
| 81 | MOD 010 Reg/Mem | Disp\_Lo, Disp\_Hi  Data Lo, Data Hi | Reg16/Mem16, Imm16 |
| 83 | MOD 010 Reg/Mem | Disp\_Lo, Disp\_Hi  Data SX | Reg16/Mem16, Imm8 |

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

**INC – INCrement operand by 1 – увеличение операнда на 1.**

|  |  |  |  |
| --- | --- | --- | --- |
| **16-ричный код (1 байт)** | **MOD Reg/OPC Reg/Mem (2-ой байт)** | **смещение**  **disp\_Lo, disp\_Hi** | **формат операндов:**  **приемник, источник** |
| FE | MOD 000 Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8/Mem8 |
| FF | MOD 000 Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16/Mem16 |

**DEC – DECrement operand by 1 – уменьшение операнда на 1.**

Коды те же (см. INC), но расширение кода операции при работе с непосредственным операндом равно 001, размещается в поле Reg/Opc в байте адресации.

**DAA – Decimel Adjust AL for Addition – десятичная коррекция в регистре AL результата сложения (**используется после команд ADD, INC**) двух упакованных BCD-чисел с целью правильного получения десятичного числа.**

Машинный код: **27h**.

**SUB – SUBtract – вычитание целых чисел.**

|  |  |  |  |
| --- | --- | --- | --- |
| **16-ричный код (1 байт)** | **MOD Reg/OPC Reg/Mem (2-ой байт)** | **смещение**  **disp\_Lo, disp\_Hi** | **формат операндов:**  **приемник, источник** |
| 28 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8/Mem8, Reg8 |
| 29 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16/Mem16, Reg16 |
| 2A | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8, Reg8/Mem8 |
| 2B | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16, Reg16/Mem16 |
| 2C | Data8 (непосредств.операнд) | отсутствует | AL, Immed8 |
| 2D | Data16 (непосред.операнд) | отсутствует | AX, Immed16 |
| 80 | MOD 101 Reg/Mem | Disp\_Lo, Disp\_Hi  Data Lo | Reg8/Mem8, Immed8 |
| 81 | MOD 101 Reg/Mem | Disp\_Lo, Disp\_Hi  Data Lo, Data Hi | Reg16/Mem16, Imm16 |
| 83 | MOD 101 Reg/Mem | Disp\_Lo, Disp\_Hi  Data SX | Reg16/Mem16, Imm8 |

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

**SBB – integer SuBtraction with Borow – вычитание с заемом.**

|  |  |  |  |
| --- | --- | --- | --- |
| **16-ричный код (1 байт)** | **MOD Reg/OPC Reg/Mem (2-ой байт)** | **смещение**  **disp\_Lo, disp\_Hi** | **формат операндов:**  **приемник, источник** |
| 18 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8/Mem8, Reg8 |
| 19 | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16/Mem16, Reg16 |
| 1A | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8, Reg8/Mem8 |
| 1B | MOD Reg/OPC Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16, Reg16/Mem16 |
| 1C | Data8 (непосредств.операнд) | отсутствует | AL, Immed8 |
| 1D | Data16 (непосред.операнд) | отсутствует | AX, Immed16 |
| 80 | MOD 011 Reg/Mem | Disp\_Lo, Disp\_Hi  Data Lo | Reg8/Mem8, Immed8 |
| 81 | MOD 011 Reg/Mem | Disp\_Lo, Disp\_Hi  Data Lo, Data Hi | Reg16/Mem16, Imm16 |
| 83 | MOD 011 Reg/Mem | Disp\_Lo, Disp\_Hi  Data SX | Reg16/Mem16, Imm8 |

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

**MUL операнд**

|  |  |  |  |
| --- | --- | --- | --- |
| **16-ричный код (1 байт)** | **MOD Reg/OPC Reg/Mem (2-ой байт)** | **смещение**  **disp\_Lo, disp\_Hi** | **формат операндов:**  **приемник, источник** |
| F6 | MOD 100 Reg/Mem | Disp\_Lo, Disp\_Hi | Reg8/Mem8 |
| F7 | MOD 100 Reg/Mem | Disp\_Lo, Disp\_Hi | Reg16/Mem16 |

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

**DIV операнд**

Коды те же (см. MUL), но расширение кода операции при работе с непосредственным операндом равно 110, размещается в поле Reg/Opc в байте адресации.

**NEG операнд**

Коды те же (см. MUL), но расширение кода операции при работе с непосредственным операндом равно 011, размещается в поле Reg/Opc в байте адресации.