Режимы адресации

Режимы адресации — это различные способы указания местоположения операндов. До этого использовались только простые режимы адресации: операнды чаще всего находились в регистрах или в переменных в памяти. Но в процессоре Intel 8086 существуют также более сложные режимы, которые позволяют организовать работу с массивами, структурами, локальными переменными и указателями. В этой работе рассмотрим возможные режимы адресации и примеры их использования.

1. Неявная адресация

Местоположение операнда фиксировано и определяется кодом операции.

Примеры:

```
cbw
mul al
```

Команда CBW всегда работает с регистрами AX и AL, а у команды MUL фиксировано положение первого множителя и результата. Такой режим адресации делает машинную команду короткой, так как в ней отсутствует указание одного или нескольких операндов.

2. Непосредственная адресация

При непосредственной адресации значение операнда является частью машинной команды. Понятно, что в этом случае операнд представляет собой константу.

Примеры:

```
mov al,5
add bx,1234h
mov dx,a
```

Обратите внимание, что в третьей строке в DX помещается *адрес* метки или переменной *a*, а вовсе не значение по этому адресу. Это особенность синтаксиса FASM. По сути адрес метки тоже является числовой константой.

3. Абсолютная прямая адресация

В машинной команде содержится адрес операнда, находящегося в памяти.

Пример:

```
mov dx,[a]
```

Вот тут уже в DX помещается значение из памяти по адресу *a*. Сравните с предыдущим пунктом. Квадратные скобки обозначают обращение по адресу, указанному внутри этих скобок.

4. Относительная прямая адресация

Этот режим используется в командах передачи управления. В машинной команде содержится смещение, которое прибавляется к значению указателя команд IP. То есть указывается не сам адрес перехода, а на сколько байтов вперёд или назад надо перейти.

Пример:

```
metka:
    ...
    loop metka
```

У такого режима адресации два преимущества. Во-первых, машинная команда становится короче, так она содержит не полный адрес, а только смещение. Во-вторых, такой код не зависит от адреса, по которому он размещается в памяти.

5. Регистровая адресация

Операнд находится в регистре.

Пример:

```
add ax,bx
```

6. Косвенная регистровая (базовая) адресация

Адрес операнда находится в одном из регистров BX, SI или DI.

Примеры:

```
add ax,[bx]
mov dl,[si]
```

Размер операнда в памяти здесь определяется размером первого операнда. Так как AX — 16-разрядный регистр, то из памяти берётся слово по адресу в BX. Так как DL — 8- разрядный регистр, то из памяти берётся байт по адресу в SI. Это правило верно и для других режимов адресации.

7. Косвенная регистровая (базовая) адресация со смещением

Адрес операнда вычисляется как сумма содержимого регистра BX, BP, SI или DI и 8- или 16-разрядного смещения.

Примеры:

```
add ax,[bx+2]
mov dx,[array1+si]
```

В качестве смещения можно указать число или адрес метки. О размере смещения не беспокойтесь — компилятор сам его определяет и использует нужный формат машинной команды.

8. Косвенная базовая индексная адресация

Адрес операнда вычисляется как сумма содержимого одного из базовых регистров BX или BP и одного из индексных регистров SI или DI.

Примеры:

```
mov ax,[bp+si]
add ax,[bx+di]
```

Например, в одном из регистров может находиться адрес начала массива в памяти, а в другом — смещение какого-то элемента относительно начала. А вообще, всё зависит от вашей фантазии

9. Косвенная базовая индексная адресация со смещением

Адрес операнда вычисляется как сумма содержимого одного из базовых регистров BX или BP, одного из индексных регистров SI или DI и 8- или 16-разрядного смещения.

Примеры:

```
mov al,[bp+di+5]
mov bl,[array2+bx+si]
```

Пример программы

Допустим, имеется массив 32-битных целых чисел со знаком. Количество элементов массива хранится в 16-битной переменной без знака. Требуется вычислить среднее арифметическое элементов массива и сохранить его в 32-битной переменной со знаком. Здесь намеренно использованы разные режимы адресации, хотя тоже самое можно написать проще.

```
use16                ;Генерировать 16-битный код
org 100h             ;Программа начинается с адреса 100h
  sub ax,ax          ;AX = 0
  cwd                ;DX = 0
  mov si,ax          ;SI = 0 - смещение элемента от начала массива
  mov bx,array       ;Помещаем в BX адрес начала массива
  mov di,n           ;Помещаем в DI адрес n
  mov cx,[di]        ;CX = n
lp1:
  add ax,[bx+si]     ;Прибавление младшего слова
  adc dx,[bx+si+2]   ;Прибавление старшего слова
  add si,4           ;Увеличиваем смещение в SI на 4
  loop lp1           ;Команда цикла
  idiv word[di]      ;Делим сумму на количество элементов
  cwd                ;DX:AX = AX
  mov word[m],ax     ;\ Сохраняем
  mov word[m+2],dx   ;/ результат
mov ax,4C00h         ;\
int 21h              ;/ Завершение программы
;---------------------------------------
n      dw 10
array  dd 10500,-7500,-15000,10000,-8000
       dd 6500,11500,-5000,10500,-20000
m      dd ?
```

Задание.

Объявите в программе два массива 16-битных целых со знаком. Количество элементов массивов должно быть одинаковым и храниться в 8-битной переменной без знака. Требуется из последнего элемента второго массива вычесть первый элемент первого, из предпоследнего — вычесть второй элемент и т.д.