

**Напиши код для masm, который будет создавать массив чисел без знака и длиной в 16 разрядов, а после будет находить сумму чисел массива с кодовой двоичной комбинацией, равной "1101"**

Для решения задачи создадим программу на MASM, которая формирует массив 16-разрядных беззнаковых чисел и вычисляет сумму элементов с младшими 4 битами 1101 (0Dh).

**Код программы:**

.model tiny  
.code  
org 100h  
  
start:  
 ; Объявление массива из 5 элементов (пример)  
 array dw 13, 5, 29, 14, 0Dh ; 13 (0x000D), 29 (0x001D) подходят  
  
 mov cx, 5 ; Количество элементов  
 xor bx, bx ; Сумма (обнуление BX)  
 mov si, offset array ; Указатель на начало массива  
  
loop\_start:  
 lodsw ; Загрузка элемента в AX  
 and ax, 0Fh ; Изоляция младших 4 битов  
 cmp ax, 0Dh ; Проверка на 1101  
 jne skip ; Пропуск, если не совпадает  
 add bx, [si-2] ; Добавление исходного значения к сумме  
  
skip:  
 loop loop\_start ; Цикл по элементам  
  
 ; Сохранение результата в переменной (опционально)  
 mov sum, bx  
  
 ; Завершение программы  
 mov ax, 4C00h  
 int 21h  
  
.data  
sum dw ?  
end start

**Пояснение:**

1. **Объявление массива**:  
   Директива dw инициализирует массив 16-разрядными значениями. Пример включает числа 13 (0x000D) и 29 (0x001D), которые удовлетворяют условию.
2. **Проверка битовой комбинации**:
   * AND AX, 0Fh изолирует младшие 4 бита.
   * CMP AX, 0Dh сравнивает результат с шаблоном 1101.
3. **Суммирование**:  
   Если условие выполняется, исходное значение (до маскирования) добавляется к сумме в регистре BX.
4. **Особенности**:
   * Модель tiny и org 100h обеспечивают совместимость с форматом COM-файла для DOS.
   * Для работы с большими массивами измените значение в CX и добавьте элементы в array.

**Примеры подходящих чисел:**

|  |  |  |
| --- | --- | --- |
| Число (DEC) | Двоичное представление | Соответствие |
| 13 | ...00001101 | Да |
| 29 | ...00011101 | Да |
| 45 | ...00101101 | Да |

Для адаптации под конкретную задачу замените значения в массиве и отрегулируйте длину цикла[[1]](#fn1)[[2]](#fn2).

⁂

**Исправленный код с поиском "1101" в любом месте 16-битного числа:**

.model tiny  
.code  
org 100h  
  
start:  
 ; Пример массива (можно добавить тестовые случаи)  
 array dw 13, 208, 29, 0ABCDh, 0D0Dh  
   
 mov cx, 5 ; Количество элементов  
 xor bx, bx ; Обнуление суммы  
 mov si, offset array ; Указатель на массив  
  
loop\_start:  
 lodsw ; Загружаем элемент в AX  
 mov di, ax ; Сохраняем исходное значение  
 push cx ; Сохраняем счётчик элементов  
 xor dx, dx ; Счётчик сдвигов (0-12)  
  
check\_pattern:  
 mov ax, di ; Восстанавливаем число  
 mov cl, dl ; Задаём количество сдвигов  
 shr ax, cl ; Сдвигаем биты  
 and ax, 0Fh ; Изолируем 4 бита  
 cmp ax, 0Dh ; Проверяем на 1101  
 je found ; Совпадение найдено  
 inc dx ; Увеличиваем сдвиг  
 cmp dx, 13 ; Проверили все позиции?  
 jb check\_pattern ; Нет -> продолжаем  
 jmp no\_match ; Пропускаем элемент  
  
found:  
 add bx, di ; Добавляем к сумме  
  
no\_match:  
 pop cx ; Восстанавливаем счётчик  
 loop loop\_start ; Обрабатываем следующий элемент  
  
 ; Сохранение результата (опционально)  
 mov sum, bx  
   
 ; Завершение программы  
 mov ax, 4C00h  
 int 21h  
  
.data  
sum dw ?  
end start

**Основные изменения:**

1. **Расширенная проверка**:

shr ax, cl ; Последовательный сдвиг на 0-12 бит  
and ax, 0Fh ; Проверка каждых 4-х битов в числе

Теперь проверяются все возможные 4-битные комбинации подряд идущих битов.

1. **Добавлен внутренний цикл**:
   * Счётчик DX для отслеживания сдвигов (0-12)
   * 13 итераций проверки для каждого числа
2. **Примеры подходящих чисел**:

|  |  |  |
| --- | --- | --- |
| Число (HEX) | Двоичное представление | Позиции с 1101 |
| 000D | 00000000 00001101 | Биты 0-3 |
| 00D0 | 00000000 11010000 | Биты 4-7 |
| ABCD | 10101011 11001101 | Биты 0-3 и 8-11 |

**Особенности реализации:**

1. **Сохранение регистров**:
   * PUSH CX/POP CX защищают счётчик основного цикла
   * DI хранит оригинальное значение числа
2. **Оптимизация**:
   * Выход из внутреннего цикла при первом совпадении
   * Минимальные накладные расходы (13 проверок/число)

Для тестирования добавьте числа с разными позициями "1101" и проверьте результат в отладчике.

**что значит "org 100h" в начале кода**

Директива ORG 100h в начале кода ассемблера указывает ассемблеру, что размещение кода и данных в памяти должно начинаться с адреса смещения 100h (256 в десятичной системе) от начала сегмента. Это связано с особенностями запуска программ типа COM в DOS, где первые 256 байт памяти (адреса 0x0000–0x00FF) зарезервированы под PSP (Program Segment Prefix) — служебный блок операционной системы.

Таким образом, ORG 100h сообщает, что реальный код программы должен располагаться сразу после PSP, начиная с адреса 100h, чтобы не перезаписывать эти служебные данные и корректно работать в среде DOS. Все метки и адреса внутри программы будут отсчитываться от этого смещения.

Это обязательная директива для COM-программ, так как DOS загружает их именно с таким смещением[[3]](#fn3)[[4]](#fn4)[[5]](#fn5)[[6]](#fn6).

⁂

**а если бы это была запись "stack 100h"**

Директива stack 100h в MASM используется для определения размера сегмента стека, а не для задания адреса или смещения.

**Что значит stack 100h?**

* **stack** — это директива, которая резервирует область памяти под стек программы.
* **100h** — размер, выделяемый под стек, в шестнадцатеричном формате (256 байт).

То есть запись

stack 100h

означает, что для стека выделяется 256 байт памяти.

**В чем разница между org 100h и stack 100h?**

|  |  |  |
| --- | --- | --- |
| Директива | Назначение | Пример использования |
| org 100h | Устанавливает начальное смещение кода или данных в сегменте (обычно для COM-программ) | Указывает, что код начинается с адреса 0x100 |
| stack 100h | Определяет размер сегмента стека (сколько памяти выделить под стек) | Резервирует 256 байт для стека |

**Пример использования stack:**

.model small  
.stack 100h ; выделяем 256 байт под стек  
.data  
; данные  
.code  
; код программы  
end

В этом примере компоновщик создаст сегмент стека размером 256 байт.

**Итог**

* org 100h — задаёт смещение начала кода/данных в сегменте.
* stack 100h — задаёт размер стека (обычно в байтах).

1. <http://forum.oszone.net/thread-177215.html>

1. <https://ravesli.com/assembler-massivy/>

1. <https://qna.habr.com/q/70174>

1. <https://av-assembler.ru/asm/afd/start-asm.htm>

1. <https://asmbase.ru/uchebnyj-kurs/002-pervaya-programma/>

1. <https://www.cyberforum.ru/asm-beginners/thread2183238.html>