数组

数组的识别

1. 有变量或变量表达式作为下标访问
    - 比例因子寻址或等价于该寻址方式的算法表达式
2. 有循环结构去处理数组内各个元素的流程
3. 高版本多媒体指令集

```
; xmm系列寄存器是128位
movaps xmm0, xxx
movups ary, xmm0
```

注意：

在高版本的Release中，可能采用指针进行寻址

`nop xxx`、`mov reg, reg`、`xor reg, reg`等等，等价于`nop`的指令一般用来使循坏在模4的地址上对齐

一维数组

1. 初始化
    - 挨个赋值

```
    int arr[] = { 47, 74, 33, 21, 40, 6, 69, 82, 29, 4 };
00211748 C7 45 D4 2F 00 00 00 mov         dword ptr [arr],2Fh
0021174F C7 45 D8 4A 00 00 00 mov         dword ptr [ebp-28h],4Ah
00211756 C7 45 DC 21 00 00 00 mov         dword ptr [ebp-24h],21h
0021175D C7 45 E0 15 00 00 00 mov         dword ptr [ebp-20h],15h
00211764 C7 45 E4 28 00 00 00 mov         dword ptr [ebp-1Ch],28h
0021176B C7 45 E8 06 00 00 00 mov         dword ptr [ebp-18h],6
00211772 C7 45 EC 45 00 00 00 mov         dword ptr [ebp-14h],45h
00211779 C7 45 F0 52 00 00 00 mov         dword ptr [ebp-10h],52h
00211780 C7 45 F4 1D 00 00 00 mov         dword ptr [ebp-0Ch],1Dh
00211787 C7 45 F8 04 00 00 00 mov         dword ptr [ebp-8],4
```

    - 使用多媒体指令集配合赋值

```
   int arr[] = { 47, 74, 33, 21, 40, 6, 69, 82, 29, 4 };
0DB1050 0F 28 05 00 21 DB 00 movaps       xmm0,xmmword ptr ds:[00DB2100h]
0DB1057 56                   push         esi
0DB1058 8B 75 08             mov          esi,dword ptr [ebp+8]
0DB105B 0F 11 45 D4          movups       xmmword ptr [ebp-2Ch],xmm0


   printf("%d\n", arr[2]);
0DB105F 6A 21                push         21h
0DB1061 0F 28 05 10 21 DB 00 movaps       xmm0,xmmword ptr ds:[00DB2110h]
0DB1068 68 F0 20 DB 00       push         0DB20F0h
0DB106D 0F 11 45 E4          movups       xmmword ptr [ebp-1Ch],xmm0
0DB1071 C7 45 F4 1D 00 00 00 mov          dword ptr [ebp-0Ch],1Dh
0DB1078 C7 45 F8 04 00 00 00 mov          dword ptr [ebp-8],4
0DB107F E8 6C 00 00 00       call         00DB10F0
```

2. 访问

- 按下标寻址公式进行访问 `type arr[index]`

$$address + sizeof(type) * index$$

```
    printf("%d\n", arr[2]);
0021178E B8 04 00 00 00       mov         eax,4
00211793 D1 E0                shl         eax,1
00211795 8B 4C 05 D4          mov         ecx,dword ptr [ebp+eax-2Ch]
00211799 51                   push        ecx
0021179A 68 D4 7B 21 00       push        217BD4h
0021179F E8 E0 FB FF FF       call        00211384
002117A4 83 C4 08             add         esp,8


    printf("%d\n", arr[argc]);
002117A7 8B 45 08             mov         eax,dword ptr [ebp+8]
002117AA 8B 4C 85 D4          mov         ecx,dword ptr [ebp+eax*4-2Ch]
002117AE 51                   push        ecx
002117AF 68 D4 7B 21 00       push        217BD4h
002117B4 E8 CB FB FF FF       call        00211384
002117B9 83 C4 08             add         esp,8

    printf("%d\n", arr[argc % 8]);
002117BC 8B 45 08             mov         eax,dword ptr [ebp+8]
002117BF 25 07 00 00 80       and         eax,80000007h
002117C4 79 05                jns         002117CB
002117C6 48                   dec         eax
002117C7 83 C8 F8             or          eax,0FFFFFFF8h
002117CA 40                   inc         eax
002117CB 8B 4C 85 D4          mov         ecx,dword ptr [ebp+eax*4-2Ch]
002117CF 51                   push        ecx
002117D0 68 D4 7B 21 00       push        217BD4h
002117D5 E8 AA FB FF FF       call        00211384
002117DA 83 C4 08             add         esp,8
```

ebp - 2ch + index * 4

ebp - 2ch + argc * 4

eax = argc % 8

ebp - 2ch + eax * 4

二维数组

1. 初始化

- 挨个赋值（同一维）

```
    int arr[3][10] = {
        { 47, 74, 33, 21, 40, 6, 69, 82, 29, 4 },
00324F88 C7 45 84 2F 00 00 00 mov         dword ptr [ebp-7Ch],2Fh
00324F8F C7 45 88 4A 00 00 00 mov         dword ptr [ebp-78h],4Ah
00324F96 C7 45 8C 21 00 00 00 mov         dword ptr [ebp-74h],21h
00324F9D C7 45 90 15 00 00 00 mov         dword ptr [ebp-70h],15h
00324FA4 C7 45 94 28 00 00 00 mov         dword ptr [ebp-6Ch],28h
00324FAB C7 45 98 06 00 00 00 mov         dword ptr [ebp-68h],6
    int arr[3][10] = {
        { 47, 74, 33, 21, 40, 6, 69, 82, 29, 4 },
00324FB2 C7 45 9C 45 00 00 00 mov         dword ptr [ebp-64h],45h
00324FB9 C7 45 A0 52 00 00 00 mov         dword ptr [ebp-60h],52h
```

- 使用多媒体指令集配合赋值（同一维）

2. 访问

- 按下标寻址公式进行访问 `type arr[N][M]` 访问 `type arr[i][j]`

$$address + sizeof(type[M]) * i + sizeof(type) * j$$

$$= address + sizeof(type) * M * i + sizeof(type) * j$$

即 $$address + sizeof(type) * (M * i + j)$$