

Scaling Array Indexes

Updated 1/3/2003

Copyright Kip R. Irvine, 2003. This article is written primarily for users of Assembly Language for Intel-Based Computers, 4th Edition. You may copy and print the article, as long as you do not alter its content.

IA-32 processors permit the offset of a memory operand to include a *scale factor* (2, 4, or 8) that is multiplied by the operand's index value.

Scale factors are particularly valuable when stepping through arrays of words, doublewords, or quadwords because they offer the same advantages as array subscripts in high-level languages. You simply increment your array index by 1 during each loop repetition, regardless of the array's type.

For example, if we were to loop through an array of doublewords pointed to by ESI, the basic code (without scaling) to sum the words would be:

```
L1: add  eax,[esi]           ; add this doubleword
      add  esi,TYPE DWORD   ; point to next element
      loop L1
```

Instead, if we use a scale factor to address each doubleword, the index needs only to be incremented by 1:

```
L1: add  eax,[esi*4]       ; add this doubleword (scaled)
      inc  esi             ; point to next element
      loop L1
```

To make the code slightly more self-documenting, we can use the TYPE operator:

```
L1: add  eax,[esi*TYPE DWORD] ; add this doubleword (scaled)
      inc  esi                 ; point to next element
      loop L1
```

When indexed addressing is used, you can use the array name when calculating the scale factor:

```
.data
myArray DWORD 500 DUP(?)
.code
mov  esi,0
```

```
    mov  ecx,LENGTHOF myArray
    mov  eax,0
L1:  add  eax,myArray[esi*TYPE myArray]
      inc  esi
      loop L1
```

An advantage to this approach is that if the array type ever changes, the loop code needs only minimal modification.