

Lecture 22: IL for Arrays

One-dimensional Arrays

- How do we process retrieval from and assignment to $x[i]$, for an array x ?
- We assume that all items of the array have fixed size— S bytes—and are arranged sequentially in memory (the usual representation).
- Easy to see that the address of $x[i]$ must be

$$\&x + S \cdot i,$$

where $\&x$ is intended to denote the address of the beginning of x .

- Generically, we call such formulae for getting an element of a data structure *access algorithms*.
- The IL might look like this:

```
cgen(&A[E], t0):  
  cgen(&A, t1)  
  cgen(E, t2)  
  ⇒ t3 := t2 * S  
  ⇒ t0 := t1 + t3
```

Multi-dimensional Arrays

- A 2D array is a 1D array of 1D arrays.
- Java uses arrays of pointers to arrays for >1D arrays.
- But if row size constant, for faster access and compactness, may prefer to represent an $M \times N$ array as a 1D array of 1D rows (not pointers to rows): *row-major order*...
- Or, as in FORTRAN, a 1D array of 1D columns: *column-major order*.
- So apply the formula for 1D arrays repeatedly—first to compute the beginning of a row and then to compute the column within that row:

$$\&A[i][j] = \&A + i \cdot S \cdot N + j \cdot S$$

for an M -row by N -column array, where S , again, is the size of an individual element.

IL for $M \times N$ 2D array

```
cgen(&e1[e2,e3], t):  
  cgen(e1, t1); cgen(e2,t2); cgen(e3,t3)  
  cgen(N, t4) # (N need not be constant)  
  ⇒ t5 := t4 * t2  
  ⇒ t6 := t5 + t3  
  ⇒ t7 := t6 * S  
  ⇒ t := t7 + t1
```

Array Descriptors

- Calculation of element address $\&e1[e2, e3]$ has the form

$$VO + S1 \times e2 + S2 \times e3$$

, where

- VO ($\&e1[0,0]$) is the *virtual origin*.
 - $S1$ and $S2$ are *strides*.
 - All three of these are constant throughout the lifetime of the array (assuming arrays of constant size).
- Therefore, we can package these up into an *array descriptor*, which can be passed in lieu of the array itself, as a kind of "fat pointer" to the array:

$\&e1[0][0]$	$S \times N$	S
--------------	--------------	-----

Array Descriptors (II)

- Assuming that $e1$ now evaluates to the address of a 2D array descriptor, the IL code becomes:

```
cgen(&e1[e2,e3], t):  
  cgen(e1, t1); cgen(e2,t2); cgen(e3,t3)  
  => t4 := *t1;      # The VO  
  => t5 := *(t1+4)  # Stride #1  
  => t6 := *(t1+8)  # Stride #2  
  => t7 := t5 * t2  
  => t8 := t6 * t3  
  => t9 := t4 + t7  
  => t10:= t9 + t8
```

Array Descriptors (III)

- By judicious choice of descriptor values, can make the same formula work for different kinds of array.
- For example, if lower bounds of indices are 1 rather than 0, must compute address

$$\&e[1,1] + S1 \times (e2-1) + S2 \times (e3-1)$$

- But some algebra puts this into the form

$$VO' + S1 \times e2 + S2 \times e3$$

where

$$VO' = \&e[1,1] - S1 - S2 = \&e[0,0] \text{ (if it existed).}$$

- So with the descriptor

VO'	$S \times N$	S
-------	--------------	-----

we can use the same code as on the last slide.