Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

02 – Arrays

CS1022 – Introduction to Computing II

Dr Jonathan Dukes / jdukes@scss.tcd.ie
School of Computer Science and Statistics

Arrays

Nothing very new here … we have already been using arrays!

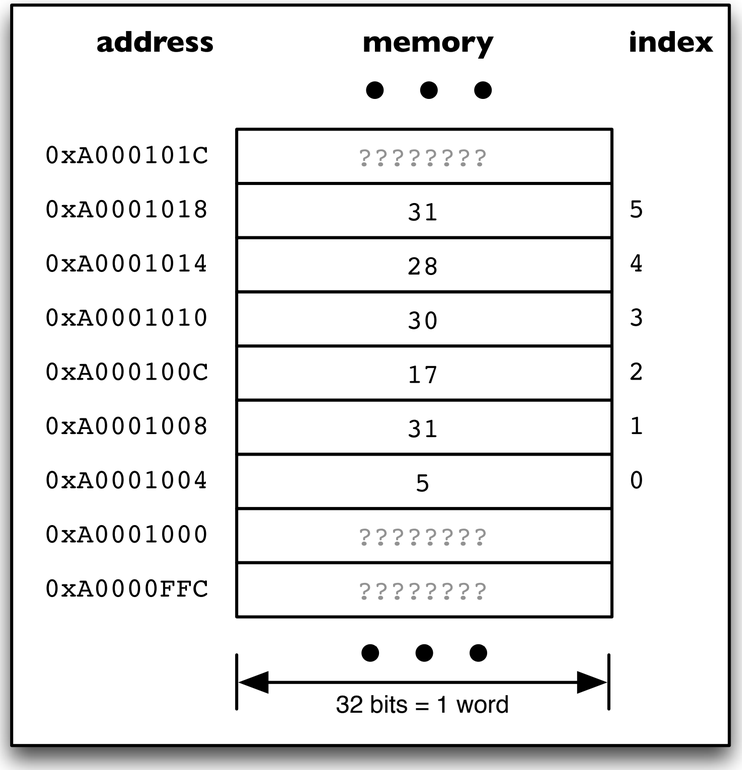
Array – an ordered collection of elements stored sequentially in memory

e.g. integers, ASCII characters, lottery numbers

Homogeneous elements?

(at least with respect to size)

Dimension: number of elements in array



Accessing (loading/storing) Array Elements

Step 1: translate array index into byte offset from start address of array in memory

<byte offset> = <index> × <elem size>

Step 2: add byte offset to array base address to access element

<address> = <array start address> + <byte offset>

Example: retrieve the 4th element (index=3) of an array of words

Efficient implementation of random access using **Scaled Register Offset** addressing mode:

```
LDR   r4, =array              ; pArr = start address of array
LDR   r5, =3                  ; index = 3 (4th element)
LDR   r6, [r4, r5, LSL #2]    ; elem = Memory.Word[pAarr + (index * 4)]
```

Array Access Example - Sum

Déjà Vu!

```
   LDR   R1, =myArray          ; start address of myArray
   LDR   R0, =0                ; sum = 0
   LDR   R4, =0                ; count = 0

whSum
   CMP   R4, #10               ; while (count < 10)
   BHS   eWhSum                ; {
   LDR   R6, [R1, R4, LSL #2]  ;  num = myArray[count]
   ADD   R0, R0, R6            ;  sum = sum + num
   ADD   R4, R4, #1            ;  count = count + 1
   B     whSum                 ; }
eWhSum                         ;
```

The pseudo-code comments have changed but the program is identical!!!

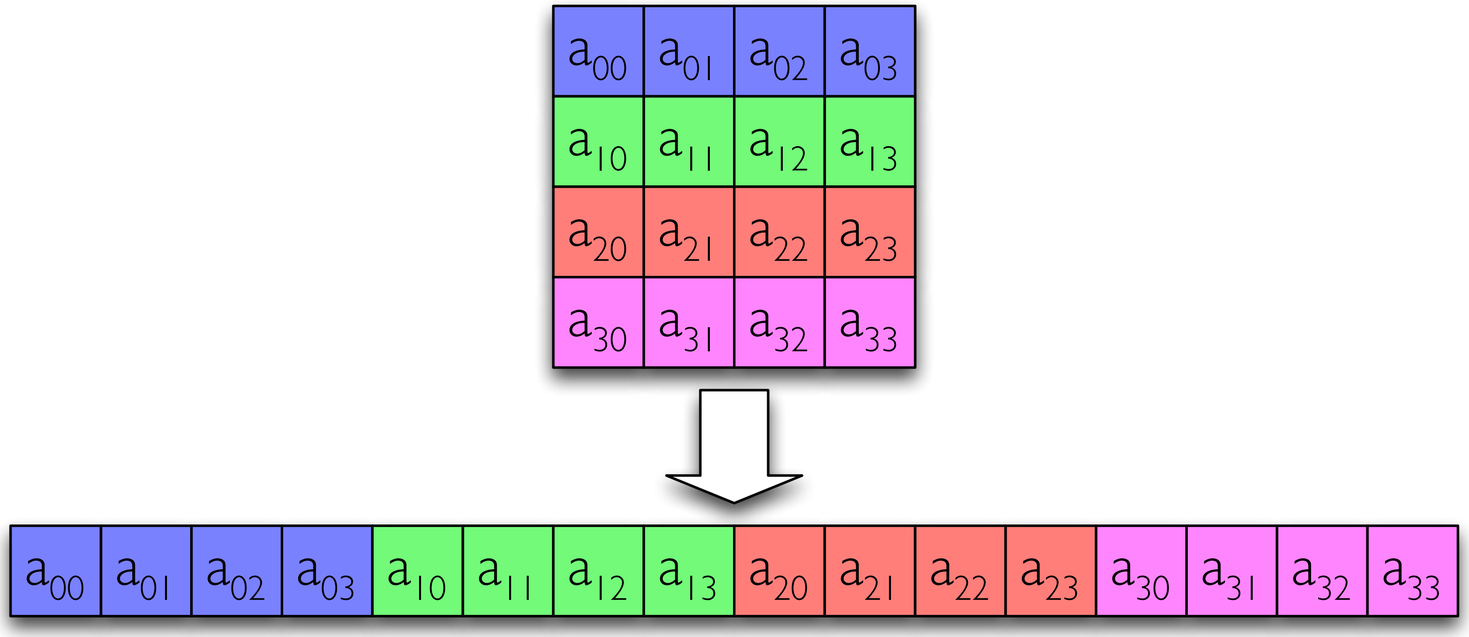
(See **Addressing Modes**)

2D Arrays

Arrays can have more than one dimension

e.g. a two-dimensional array – analogous to a table containing elements arranged in rows and columns

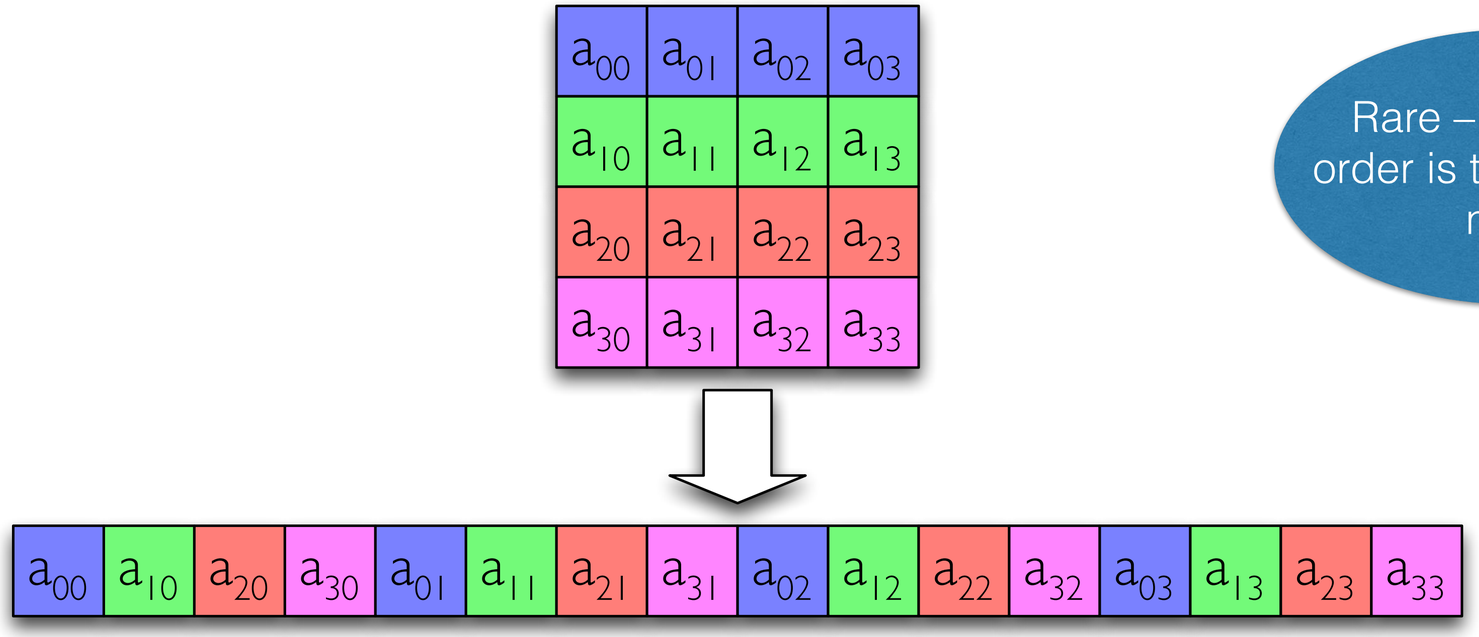
Stored in memory by mapping the 2D array into 1D memory, e.g.



2D Arrays

Row-major order: 2D array is stored in memory by storing each **row** contiguously in memory

Column-major order: 2D array is stored in memory by storing each **column** contiguously in memory



Rare – row-major order is the assumed norm

2D Array Example

2D array declared in memory

```
        AREA  TestData, DATA, READWRITE

array   DCD   6, 3, 8, 2, 5, 2, 9, 1 ; row 0
        DCD   3, 7, 2, 8, 5, 7, 2, 7 ; row 1
        DCD   2, 4, 7, 4, 2, 6, 7, 4 ; row 2
        DCD   1, 9, 3, 2, 9, 5, 6, 8 ; row 3
        DCD   7, 5, 3, 7, 5, 8, 2, 1 ; row 4
        DCD   6, 4, 8, 9, 0, 3, 2, 5 ; row 5
```

… or equivalently …

Why are these equivalent?

```
        AREA  TestData, DATA, READWRITE

array   DCD   6, 3, 8, 2, 5, 2, 9, 1, 3, 7, 2, 8, 5, 7, 2, 7, 2, 4, 7, 4, 2
        DCD   6, 7, 4, 1, 9, 3, 2, 9, 5, 6, 8, 7, 5, 3, 7, 5, 8, 2, 1, 6, 4
        DCD   8, 9, 0, 3, 2, 5
```

Accessing 2D Array Elements

Example: retrieve the element at the 3rd row and 2nd column of a 2D array of words with 6 rows and 8 columns – **array[3][2]**

Step 1: translate 2D array index into 1D array index

<index> = (<row> × <row size>) + <col>

Step 2: translate 1D array index into byte offset from start address of array in memory

<byte offset> = <index> × <elem size>

Step 3: add byte offset to array base address to access element

<address> = <array base address> + <byte offset>

Step 1 is new

Steps 2 & 3 are the same as those for a 1-D array …

… because our 2-D array is just a different interpretation of a 1-D array!

2D Array Access Example

Example: retrieve the element at the 3rd row and 2nd column of a 2D array of words with 6 rows and 8 columns – array[3][2]

```
LDR   r4, =array              ; pArr = address of array start
LDR   r5, =col_size           ; load col_size
LDR   r6, =row_size           ; load row_size

; looking for array[3][2] (3rd row, 2nd column)

LDR   r1, =3                  ; row = 3
LDR   r2, =2                  ; col = 4

; <byte offset> = ((row * <row_size>) + col) * <elem size>

MUL   r7, r1, r6              ; index = row * row_size
ADD   r7, r7, r2              ; index = index + col
LDR   r0, [r4, r7, LSL #2] ; elem = Memory.Word[ pArr + (index*4) ]
```

index calculation

array access

Multi-dimension arrays

e.g. a 3D array of size sz×sy×sx

In general, the index of element a[z][y][x] is:

$$\textbf{index} = ((z \times sy \times sx) + (y \times sx) + x)$$

e.g. a 4D array of size sz×sy×sx×sw

In general, the index of element a[z][y][x][w] is:

$$\textbf{index} = ((z \times sy \times sx \times sw) + (y \times sx \times sw) + (x \times sw) + w)$$

Warning: an array of bytes with odd dimensions may cause the data following the array to be odd aligned, requiring padding of one byte (similarly for half-words)