Lab# 5 ARRAYS

Instructor: I Putu Danu Raharja.

Objectives:
To introduce the students how to implement array as an abstract data structure in MIPS assembly language.

Method:

Translate an algorithm from pseudo-code into assembly language.

Preparation:

Read the chapter 2 of lecture textbook.

5.1 INTRODUCTION

Not like high level languages, Assembly language has no notion of an array at all. Arrays like variables are treated as a block of memory that could be allocated with a single directive, where the first element is given a label.

The array as the most important and most general data structure has the following properties:

1. All elements must be the same size. The array is an homogeneous data structure.
2. The size of an array is fixed. The number of elements is fixed.
3. A label (address) is tied to the first element of the array.
4. Traversing each element of an array needs an index or indices and the label as the array's name.

5.2 ARRAY DECLARATION

With reference to the above properties, in assembly language to declare an array it requires:

1. A label name,
2. The number of elements,
3. The size of each element,
4. The initial value of each element.

A. Example:

```
        .data
A01:    .byte   'a', 'k', 'p', 5     # A01 is an array of 4 bytes: {'a', 'k', 'p', 5}
A02:    .word   5, 6, -9, 7          # A02 is an array of 4 words: {5, 6, -9, 7}
B02:    .space  40                   # allocate 40 consecutive bytes, with storage uninitialized
                                     # could be used as a 40-element character array, or a
                                     # 10-element integer array;
                                     # a comment should indicate which!
var1:   .half   3                    # create a single short integer variable with initial value 3
B03:    .word   -1:30                # allocate 40 consecutive words with each element
                                     # initialized with -1.
```

5.3 TRAVERSING SINGLE-DIMENSIONAL ARRAY

To access every element of an array, we have to know the address of that element. Because all elements have the same size, the address of an element of the array can be formulated as:

The address of ith-element (in byte) = starting address + ***size-of-element*** * i

1. The first element of the array is indexed 0.
2. The ***size-of-element*** is the number of bytes in a single array element.
3. The ***size-of-element*** either is one byte, 2 bytes, 4 bytes, or 8 bytes.

A. Example:

The following code fragment is to access the sixth element of table1:

```
        .data
table1: .word           4, 5, 6, 7, 8, 9, 10, 21
        .text
        la      $t0, table1
        lw      $t1, 20($t0)
        addiu   $t2, $t0, 20
        lw      $t1, 0($t2)
```

5.4 TWO-DIMENSIONAL ARRAYS

Two or higher dimensional arrays are treated as the same as simple single-dimensional arrays.

To declare the array M[rows][cols] of byte-sized elements,

1. Calculate the number of elements in the array: number-of-elements = rows * cols.

2. Then you may declare as:

   M: .byte 0:*number_of_elements*

5.5 STORAGE ORDER

As mentioned before that memory is organized as a single-dimensional array. Two-dimensional arrays must be treated as simple single-dimensional arrays. Then, in assembly language to declare two-dimensional arrays, we have to arrange the arrays as single-dimensional arrays.

To do this, we have to know how to organize all elements of an array. There are two different ways to organize the elements of two-dimensional array:

1. ***Row-major order***: The array is organized as a sequence of ROWS. Most of programming languages such as C follow this method.

2. ***Column-major order***: The array is organized as a sequence of COLUMNS. This order is being implemented in FORTRAN.

A. Address Calculation

Assume the row and column index starts from 0. The general formula to calculate the byte address of the element [a, b] can be expressed as:

Row-major order: Starting Address + Size-of-element * (a * number-of-columns + b)

Column-major order: Starting Address + Size-of-element * (b * number-of-rows + a)

Example:

Suppose the array size has 2 rows and 3 columns:

(0, 0)	(0, 1)	(0, 2)
(1, 0)	(1, 1)	(1, 2)

The array stored in row-major order:

(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)
0	n	2n	3n	4n	5n
Lower address					Higher address

The array stored in column-major order:

(0,0)	(1,0)	(0,1)	(1,1)	(0,2)	(1,2)
0	n	2n	3n	4n	5n
Lower address					Higher address

Example:

Pseudo-code	MIPS Assembly Language

int M[][] = new int[9][4];
 	
for(int j=0; j<9; j++)
M[j][3] = 7;
	.data
	M: .word 0:36
	# The size-of-element = 4 bytes.
	# a = j, b = 3, number-of-columns = 4 ; Then,
	# the offset of M[3][4] is 4 * (j * 4 + 3) = 4*(4*j) + 12
	.text
	la $t2, M
	li $t1, 9
	li $t0, 0
	li $t4, 7
	L2: beq $t0, $t1, X2
	sll $t3, $t0, 4
	addiu $t3, $t3, 12
	addu $t3, $t3, $t2
	sw $t4, 0($t3)
	addiu $t0, $t0, 1
	j L2
	X2:

Example:

The solution of above example using pointer:

```
    .data
M:  .word 0:36
    .text
        la      $t2, M
        li      $t1, 9
        li      $t0, 0
        li      $t4, 7
        addiu   $t2, $t2, 12
L2:     beq     $t0, $t1, X2
        sw      $t4, 0($t2)
        addiu   $t0, $t0, 1
        addiu   $t2, $t2, 16
        j       L2
X2:
```

5.6 EXERCISES

1. Write a MIPS assembly language to sort an integer (32 bits) array in ascending order using *insertion sort* algorithm. Use pointer technique.

2. Write a MIPS assembly language to transpose an integer matrix.