Project: Array operations in MIPS

Project: Array operations in MIPS

Objectives: The purpose of this projectt is to give you some practice thinking about algorithm development and programming in MIPS, and to help you gain a better understanding of memory addressing and usage in MIPS.

This is a pair programming assignment. Please work with your assigned partner. You only need to hand in one copy of the code, although **both of your names should appear at the top of each file you submit.** The assignment is due Wednesday, 19 September 2012 by the start of class.

Introduction and background

Array, or list, operations are common in most higher-level programming languages. Recall that an array is defined as a block of contiguous memory that stores items that the programmer deems to be related in some way, which we refer to by a single name; individual items in the array are referred to by their distance from the front of the array (their index in the array, or position). Examples of array usage include the grades on the last reading quiz, the pixels in an image, and the characters in a string.

Different programming languages have different conventions for storing the size of an array. In Java, for instance, the size of an array is a part of the array's data structure. This allows the language to do common operations like determining the size of an array quickly, rather than having to count the items in the array repeatedly.

For this assignment, you will be writing MIPS code to operate on arrays. The code you will write will implement the machine code version of array operations that are common in higher-level programming languages such as Java. You will use two mechanisms for storing arrays in MIPS. In the first, for a given array, you will set aside the first slot in that array to store the current size of the array, store the first data item in the second slot, etc. In the second, you will use a "sentinel" value (some specific number) to indicate the end of the array.

Your task

This assignment has two parts to it.

Part 1: Storing the size of the array in the array

Implement each of the array operations described below. You do not have to write these as functions---you can just write the appropriate MIPS code to accomplish the task. Make sure you label your code, using appropriate commenting style. For any task that changes the size of the array, make sure you update the size "field" as well. Put each task in ~~a separate~~ **the same file** (i.e. you should have one file with all of your code when you're finished with the project).

1. DELETE: This operation deletes the first occurrence of the specified item in the array. It then shifts all of the remaining items over. If the item is not in the array, the program should store -1 in register $s0; otherwise, it should store 1 in $s0. The value of the item to remove should be loaded into register $s1.

2. INSERT: This operation inserts an item into the specified location in the array. The value of the item to insert should be loaded into register $s2, and the position into which to insert the item should be loaded into register $s3. (Note: this position should be zero-based, even though we don't actually store any data in location 0. In other words, you should indicate relative position just as you would in a higher-level language: 0 indicates the first slot in the array, etc.) If the item cannot be inserted into the specified position (i.e., it specifies a position beyond the end of the existing array) it should be appended to the end of the

array.
3. GET: This operation retrieves the item stored at a specific location in the array, but does not alter the contents of the array. The index of the item to retrieve should be loaded into register $s5. Once retrieved, your program should place this value into register $s6.

In each case, you should store the starting address of the array in register $s4.

Part 2: Marking the end with a sentinel value

An alternate mechanism for indicating the size of an array is to include a "sentinel" value at the end of the array. For instance, C specifies the end of a string by the presence of NULL ('\0'). Choose **one** of the array operations from Part 1 and reimplement them assuming the following model:

- The first item in the array appears in slot 0, the second in slot 1, etc.
- The array is terminated by a -1 in the last position.
- There is no explicit size "field" (i.e. the size is not explicitly stored as part of the array).

In addition, implement a length() procedure that calculates the current size of the array, and returns this value to the caller.

(Note: Do not overwrite your original implementation! Your submitted code should contain both versions of this operation.)

In a **plain text file (i.e. one you'd write in TextWrangler)**, write a paragraph or 2 discussing at least one pro and one con of each implementation. Based solely on your experience implementing the same operation using both storage models, which one is preferable for your chosen operations? Briefly justify your answer.

Helpful hints

The easiest way to load constants into registers is using the pseudoinstruction li, or "load immediate". Use this instruction to load the starting address of the array into a register. (You can also use a combination of li and sw to populate your array in memory, as an alternative to manually entering each element of the array into memory.)

Please use comments to clearly indicate which section of the code is solving which problem.

Evaluation and grading checklist

This assignment is worth 30 points. Each operation is worth 5 points and the writeup is also worth 5 points. You will get full credit for each operation if it does the following:

- uses the correct registers, as specified in the problem
- maintains the correct value for SIZE (or, alternately, maintains the sentinel in the correct position in the array)
- indexes the array locations appropriately and correctly
- executes the operation, as described in the problem, correctly.

Your writeup will receive full credit if it does the following:

- correctly *and clearly* lists *and describes* one pro and one con for each implementation of the chosen operation
- provides sufficient and rational justification for "which storage model is preferable", that refers and relates back to your discussion of the pros and cons for each implementation
- is clearly and concisely written, with few to no spelling and grammar errors.

Submission instructions

All of your code should be in a single assembly file, named arrays.asm. Your writeup should be in a **plain text file** named analysis.txt. Submit both of these via Moodle, and make sure your name and your partner's name appears at the top of each file.

Grading summary