Laboratory Exercise 6

Array and Pointer

Goals

After this laboratory exercise, students should be able to understand how array and pointer are represented. Moreover, students will be able to differentiate index and pointer, which are used when looping through an array or a list.

Preparation

Students should review the textbook Computer Organisation and Design by Patterson & Henessy (Section 2.8, 2.13).

Array and Pointer

A large number programming tasks require looping through an array or a list, i.e. examining each element of the array/list in turn. For example, to determine the largest value in a list of integers, we need to examine every element of the list. There are two basic ways to accomplish this task:

1. *Index*: which uses a register for holding the index i of the list. The value in the register is incremented to move from the current element (i) of the list to the next element (i+1).
2. *Pointer*: which uses a register for holding the address of the being-examined element in the list. The value in the register is updated to point to the next element.

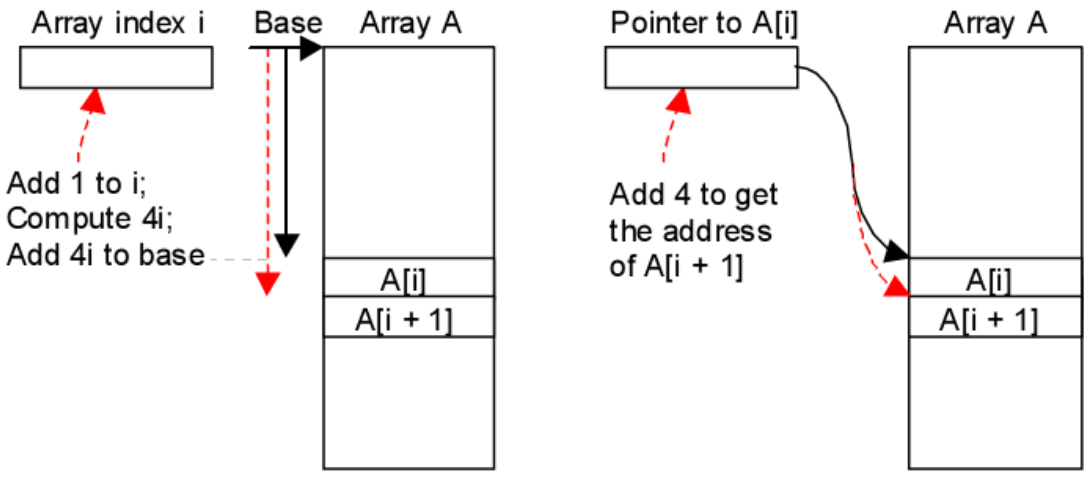


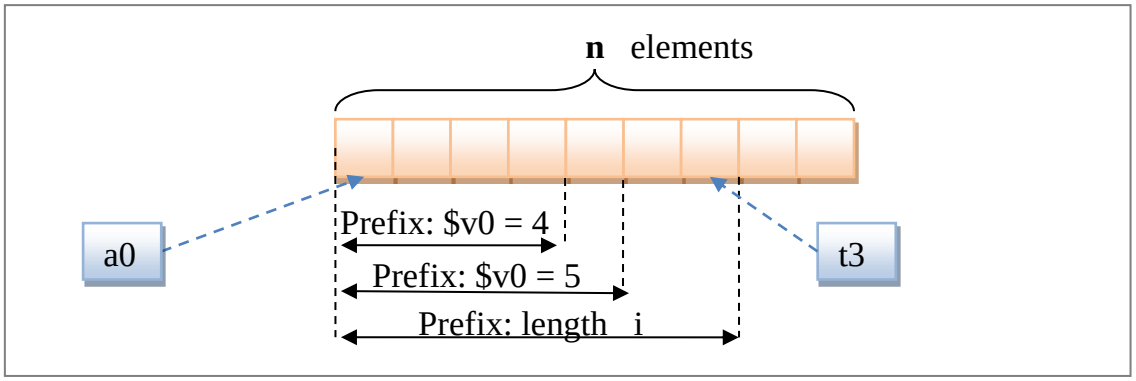
Figure 4. Using the indexing method and the pointer-updating method to loop through all elements of an array

Assignments

Sample Code 1

Given a list of n integers, a prefix list of i integers ($0 \le i \le n$) of the given list consists of the first i integers in the list; a prefix sum is the sum of all these integers. For example, if we have a list of 4 integers {2, -3, 2, 5}, the prefix sum of the prefix list of 3 integers will be 1 (2 + (-3) + 2 = 1).

The following program is used to find the maximum prefix sum of a list of 5 integers. Note that this program uses the indexing method to loop through the given list. Read this program carefully.



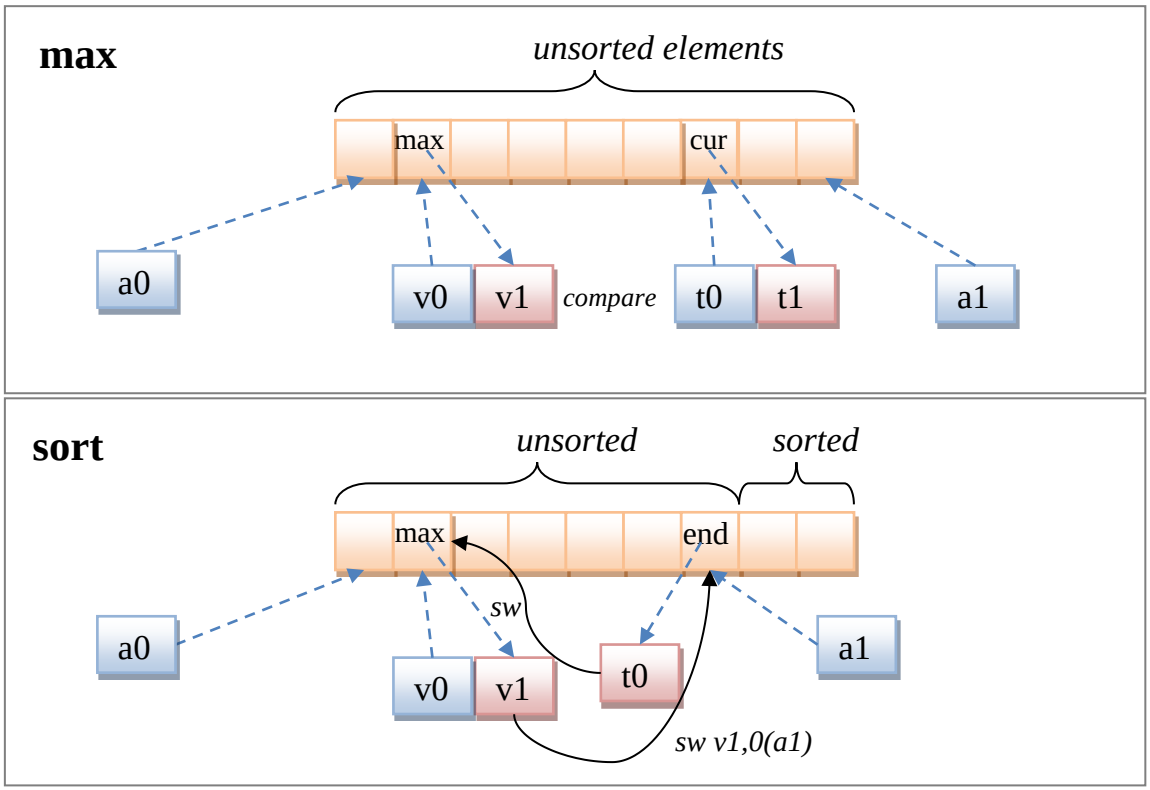

```
.data
A: .word -2, 6, -1, 3, -2

.text
main:       la    $a0,A
            li    $a1,5
            j     mspfx
            nop
continue:
lock:       j  lock
            nop
end_of_main:

#-----------------------------------------------------------------
#Procedure mspfx
# @brief find the maximum-prefix sum in a list of integers
# @param[in] a0: the base address of the list (A) need to be processed
# @param[in] a1: the number of elements in list (A)
# @param[out]v0: the length of sub-list of A that has the max sum.
# @param[out]v1: the max prefix sum
#-----------------------------------------------------------------
#Procedure mspfx
#function: find the maximum-prefix sum in a list of integers
#the base address of the list (A) is stored in $a0
#the number of elements is stored in a1
mspfx:  addi      $v0,$zero,0 #initialize the length in $v0 to 0
        addi      $v1,$zero,0 #initialize the max sum in $v1 to 0
        addi      $t0,$zero,0 #initialize the index i in $t0 to 0
        addi      $t1,$zero,0 #initialize the running sum in $t1 to 0
loop: add   $t2,$t0,$t0       #put 2i in $t2
      add   $t2,$t2,$t2       #put 4i in $t2
      add   $t3,$t2,$a0       #put 4i+A (address of A[i]) in $t3
      lw    $t4,0($t3)        #load A[i] from mem(t3) into $t4
```

```
      add   $t1,$t1,$t4       #add A[i] to the running sum in $t1
      slt   $t5,$v1,$t1       #set $t5 to 1 if max sum < new sum
      bne   $t5,$zero,mdfy    #if the max sum is less, modify results
      j     test              #done?
mdfy: addi  $v0,$t0,1         #new max-sum prefix has length i+1
      addi  $v1,$t1,0         #new max sum is the running sum
test: addi  $t0,$t0,1         #increment the index i
      slt   $t5,$t0,$a1       #set $t5 to 1 if i<n
      bne   $t5,$zero,loop    #repeat if i<n
done: j     continue
mspfx_end:
```

Sample Code 2

A given list of n numbers can be sorted in ascending order by the following algorithm (a.k.a selection sort).
1. Maintain 2 sub-lists in the given list, sorted list and unsorted list.
2. Repeatedly find the largest number (maybe more than one) in the unsorted sub-list.
3. Swap the largest number with the last element in the unsorted sub-list. Then, move that number to the sorted sub-list. Now, repeat the above steps for the remaining elements in the unsorted sub-list. When there is only one element left in the unsorted sub-list, the sorting process ends.

The following progam shows how the selection sorting algorithm is written in assembly. Note that this program uses the pointer-updating method to loop through the given list. Read this program carefully.



```
.data
A: .word 7, -2, 5, 1, 5,6,7,3,6,8,8,59,5
Aend: .word

.text
main:       la   $a0,A           #$a0 = Address(A[0])
            la   $a1,Aend
            addi $a1,$a1,-4      #$a1 = Address(A[n-1])
            j    sort            #sort
after_sort: li   $v0, 10         #exit
            syscall
end_main:
#---------------------------------------------------------------
#sort procedure (selection sort using pointer for ascending order)
#$a0: pointer to the first element in the unsorted part
#$a1: pointer to the last element in the unsorted part
#$t0: temporary place for the value of the last element
#$v0: pointer to the max element in the unsorted part
#$v1: value of the max element in the unsorted part
#---------------------------------------------------------------
sort:       beq     $a0,$a1,done    #single-element list is sorted
            j       max             #call max procedure
after_max:  lw      $t0,0($a1)      #load last element into $t0
            sw      $t0,0($v0)      #copy last element to max location
            sw      $v1,0($a1)      #copy max value to last element
            addi    $a1,$a1,-4      #decrement pointer to the last
element
            j       sort            #repeat sort for a smaller list
done:       j       after_sort

#---------------------------------------------------------------------
#Procedure max
#function: fax the value and address of max element in the list
#$a0 pointer to the first element
#$a1 pointer to the last element
#---------------------------------------------------------------------
max:
      addi  $v0,$a0,0         #initialize max pointer to first element
      lw    $v1,0($v0)        #initialize max value to first value
      addi  $t0,$a0,0         #initialize next pointer to first
loop:
      beq   $t0,$a1,ret       #if next=last, return
      addi  $t0,$t0,4         #move to next element
      lw    $t1,0($t0)        #load next element into $t1
      slt   $t2,$t1,$v1       #(next)<(max) ?
      bne   $t2,$zero,loop    #if (next)<(max), repeat
      addi  $v0,$t0,0         #next element is new max element
      addi  $v1,$t1,0         #next value is new max value
      j     loop              #change completed; now repeat
ret:
      j     after_max
```

Assignment 1

Create a new project to implement the procedure in Sample Code 1. Add more code lines to the program (if necessary) and initialize the integer list again. Run this program step by step, observe and analyze the results.

Assignment 2

Create a new project to implement the procedure in Sample Code 2. Add more code lines to the program (if necessary) and initialize the integer list again. Run this program step by step, observe and analyze the results.

Assignment 3

Write a procedure to implement the bubble sorting algorithm. Given that the list to be sorted in the ascending order consists of n integers ($20 \leq n \leq 30$).

Assignment 4

Write a procedure to implement the insertion sorting algorithm. Given that the list to be sorted in the ascending order consists of n integers ($20 \leq n \leq 30$).

Questions

- What is the advantage/disadvantage of the indexing and pointer-updating methods?