Lab 06. Array and Pointer

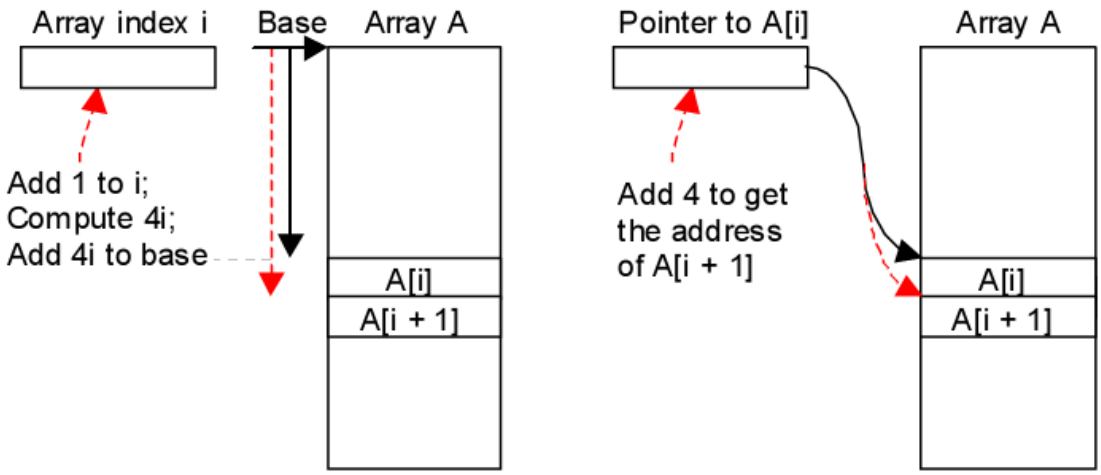
Goals

After this lab session, students will understand how arrays and pointers are represented and will be able to differentiate between using array indexing and pointer to traverse the elements of an array.

Array and Pointer

Many programming tasks involve traversing array elements. For example, finding the largest element in an array or sorting the elements of an array. There are two basic methods for traversing array elements: using array indexing or pointer.

1. Indexing: A register is used to store the value of the index i of the element to be accessed, and the index is increased or decreased to access other elements of the array.
2. Pointer: A register stores the address of the element to be accessed, and the value of the register is changed to point to other elements in the array.



Using the indexing method and the pointer updating method to step through the elements of an array.

Assignments at Home and at Lab

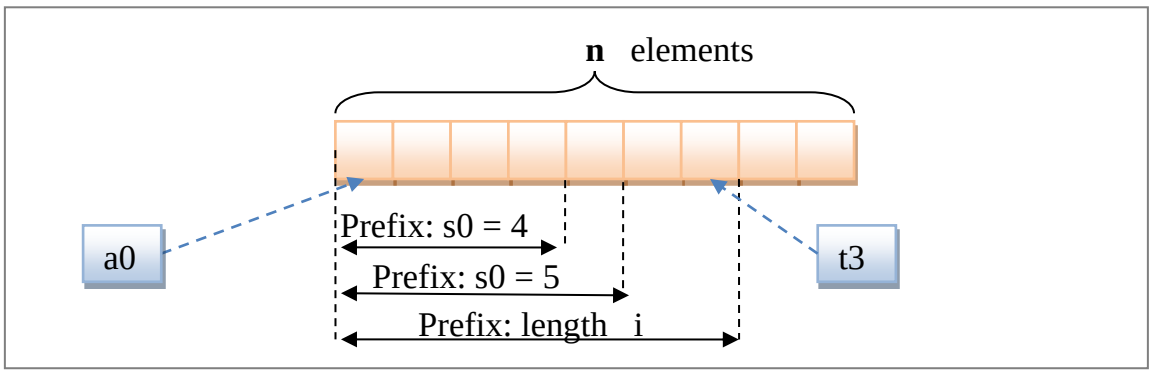
Home Assignment 1

Algorithm for finding the maximum prefix sum:

Given an array of integers with length n, the prefix of length i consists of the first i elements in the array, where $1 \leq i \leq n$. A prefix sum is the sum of the first i elements in the array. Find the maximum prefix sum of the given array.

Example: Given the array (2, -3, 2, 5, -4), the maximum prefix sum includes the first four elements, and the corresponding sum is (2 + -3 + 2 + 5) = 6.

The program below implements this algorithm using indexing-based access. Read and understand how the program works.




```
.data
A: .word -2, 6, -1, 3, -2
.text
main:
   la    a0, A
   li    a1, 5
   j     mspfx
continue:
exit:
   li    a7, 10
   ecall
end_of_main:

# ------------------------------------------------------------------
# Procedure mspfx
# @brief find the maximum-sum prefix in a list of integers
# @param[in] a0 the base address of this list(A) needs to be processed
# @param[in] a1 the number of elements in list(A)
# @param[out] s0 the length of sub-array of A in which max sum reachs.
# @param[out] s1 the max sum of a certain sub-array
# ------------------------------------------------------------------
# Procedure mspfx
# Function: find the maximum-sum prefix in a list of integers
# The base address of this list(A) in a0 and the number of
# elements is stored in a1

mspfx:
   li    s0, 0          # initialize length of prefix-sum in s0 to 0
   li    s1, 0x80000000 # initialize max prefix-sum in s1 to smallest int
   li    t0, 0          # initialize index for loop i in t0 to 0
   li    t1, 0          # initialize 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
```

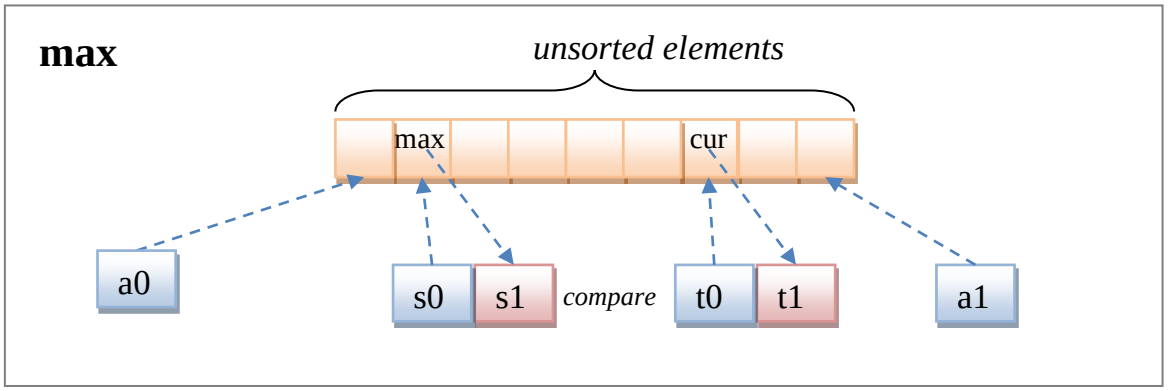
```
    lw    t4, 0(t3)      # load A[i] from mem(t3) into t4
    add   t1, t1, t4     # add A[i] to running sum in t1
    blt   s1, t1, mdfy   # if(s1 < t1) modify results
    j     next
mdfy:
    addi  s0, t0, 1      # new max-sum prefix has length i+1
    addi  s1, t1, 0      # new max sum is the running sum
next:
    addi  t0, t0, 1      # advance the index i
    blt   t0, a1, loop   # if(i<n) repeat
done:
    j     continue
mspfx_end:
```

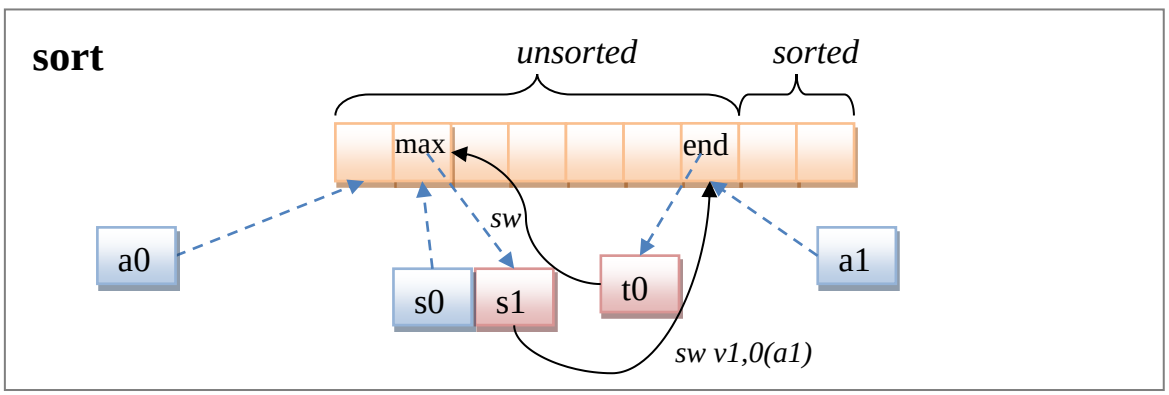
Home Assignment 2

Selection Sort Algorithm:

An array of **n** integers can be sorted in ascending order as follows: Find the largest element in the list and swap it with the last element of the array. The last element is now in its correct position. Repeat these steps for the remaining **n - 1** unsorted elements until only one element remains. At this point, the algorithm is complete, and the array is sorted in ascending order.

The program below demonstrates the implementation of the selection sort algorithm using pointer-based access. Read and understand how the program operates.





```
.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    a7, 10
   ecall
end_main:

# ------------------------------------------------------------
# Procedure sort (ascending selection sort using pointer)
# register usage in sort program
# a0 pointer to the first element in unsorted part
# a1 pointer to the last element in unsorted part
# t0 temporary place for value of last element
# s0 pointer to max element in unsorted part
# s1 value of max element in unsorted part
# ------------------------------------------------------------
sort:
   beq   a0, a1, done   # single element list is sorted
   j     max            # call the max procedure
after_max:
   lw    t0, 0(a1)      # load last element into $t0
   sw    t0, 0(s0)      # copy last element to max location
   sw    s1, 0(a1)      # copy max value to last element
   addi  a1, a1, -4     # decrement pointer to last element
   j     sort           # repeat sort for smaller list
done:
   j     after_sort

# ------------------------------------------------------------------
# Procedure max
# function: fax the value and address of max element in the list
# a0 pointer to first element
# a1 pointer to last element
# ------------------------------------------------------------------
max:
   addi  s0, a0, 0   # init max pointer to first element
   lw    s1, 0(s0)   # init max value to first value
   addi  t0, a0, 0   # init next pointer to first
loop:
   beq   t0, a1, ret # if next=last, return
   addi  t0, t0, 4   # advance to next element
```

```
    lw    t1, 0(t0)   # load next element into $t1
    blt   t1, s1, loop   # if (next)<(max), repeat
    addi  s0, t0, 0   # next element is new max element
    addi  s1, t1, 0   # next value is new max value
    j     loop        # change completed; now repeat
ret:
    j     after_max
```

Assignment 1

Create a project that implements the program in Home Assignment 1. Initialize a new set of values for the array and compile it. Run the program step by step and observe the changes in the registers to verify that the program works according to the algorithm.

Assignment 2

Create a new project that implements the program in Home Assignment 2. Initialize a new set of values for the array and compile it. Run the program step by step and observe the changes in the registers to verify that the program works according to the algorithm. Write an additional subprogram to print the array after each sorting pass.

Assignment 3

Write a program that implements the bubble sort algorithm.

Assignment 4

Write a program that implements the insertion sort algorithm.