**LAB REPORT**

**IT3280E– 152049– Assembly Language and Computer Architecture Lab**

**Lab 06: Array and Pointer**

**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.*

* Initialize a new set of values:

+ Array A = {3, 4, 8, -6, -5}

*A screenshot of a computer

Description automatically generated*

* Observe change in registers:

+ Initializing base and size of array:

* **la a0, A:** a0 is loaded with the base address of array A (a0 = 0x10010000)
* **li a1, 5:** store total number of elements in array A in *a1* (a1 = 5)
* **j mspfx:** jumps to *mspfx* procedure

**A green and grey rectangle

Description automatically generated**

**+** mspfx procedure:

* **li s0, 0:** stores length of minimum sum subarray to *s0* (s0 = 0)
* **li s1, 0x80000000:** stores the maximum sum into *s1* (right now it is initialize to the smallest possible integer/s1=- 2147483648)
* **li t0, 0:** *t0* is used as loop index (i), starting from 0 (t0 = 0)
* **li t1, 0:** stores the running sum of the prefix into *t1* (t1 = 0)

A screenshot of a video game

Description automatically generated

**+** Loop through array:

* Iteration 1:

A screenshot of a table

Description automatically generated



* Iteration 2:

A screenshot of a table

Description automatically generated



* Iteration 3:

A screenshot of a table

Description automatically generated

* Iteration 4:

A screenshot of a table

Description automatically generated



* Iteration 5:

A screenshot of a table

Description automatically generated



**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.*

* Initialize a new set of values:

+ Array A = { -3, 4, -8, -7, 5, 1, -14, -15, 6, 10}

A screenshot of a computer

Description automatically generated



* Observe change in registers:

+ Initializing base and size of array:

* **la a0, A:** store address of the first element of the array *A[0]* in *a0*
* **la a1, Aend:** loads the address of the label *Aend* into the register *a1*
* **addi a1, a1, -4:** store address of the last element of the array *A[n-1]* in *a1*
* **j sort:** jumps to *sort* procedure

****

**+** sort procedure:

* jump to max function: initialize *s0* and *s1* with the first element 
* loop through max (iterates through each element of array, comparing it to s1)

*s0: pointer to the current maximum element*

*s1: current maximum value*

*t0: holds the pointer to the element being compared*

*t1: holds the value of the element being compared*

* Each time an element greater than s1 is found, both *s0* and *s1* are updated.

*t0 is incremented to point to the next element.*

*t1 is loaded with the value of that element.*

*If t1 > s1, s0 and s1 are updated.*

A white and green rectangle with gray stripes

Description automatically generated

* after\_max function:

A screenshot of a computer

Description automatically generated

* t0: contains the value of the last element

a1: points to the last element of the unsorted part

s0: points to the current max element

s1: holds the value of the max element

* repeat sort loops

**+** Finish sorting:

* When *a0* equals *a1*, it means the array is fully sorted.
* The program jumps to *after\_sort* and exits with a system call (li a7, 10).
* Sorted array:



* Subprogram to print array:

A screenshot of a computer program

Description automatically generated

**Assignment 3:**

|  |  |
| --- | --- |
| .data  A: .word -3, 4, -8, -7, 5, 1, -14, -15  n: .word 8  newline: .asciiz "\n" # Use .asciiz instead of .ascii  .text  main:  la a0, A # a0 = address(A[0])  lw a1, n # size of array  jal bubble\_sort  la a0, A # a0 = address(A[0])  lw a1, n # reload size of array  jal print\_array  exit:  li a7, 10  ecall  # --------------------------------------------------------------  # Procedure bubble\_sort  # register usage:  # a0: pointer to base of array A  # a1: number of elements in array (n)  # t0: loop index i  # t1: loop index j  # t2: temporary variable for array value swap  # t3: element at index j  # t4: element at index j+1  # --------------------------------------------------------------  bubble\_sort:  addi t0, zero, 0 # i = 0  outer\_loop:  blt t0, a1, inner\_loop\_init # while (i < n)  j end\_sort  inner\_loop\_init:  addi t1, zero, 0 # j = 0 | inner\_loop:  sub t5, a1, t0 # n - i  addi t5, t5, -1 # n - i - 1  blt t1, t5, compare\_swap # while (j < n - i - 1)  addi t0, t0, 1 # i++  j outer\_loop  compare\_swap:  slli t2, t1, 2 # t2 = j \* 4 (byte offset for A[j])  add t3, a0, t2 # t3 = address of A[j]  lw t4, 0(t3) # t4 = A[j]  lw t5, 4(t3) # t5 = A[j+1]  bgt t4, t5, swap # if A[j] > A[j+1], swap  j next  swap:  sw t5, 0(t3) # A[j] = A[j+1]  sw t4, 4(t3) # A[j+1] = A[j]  next:  addi t1, t1, 1 # j++  j inner\_loop  end\_sort:  jr ra # return from bubble\_sort |

**Assignment 4:**

|  |  |
| --- | --- |
| .data  A: .word -3, 4, -8, -7, 5, 1, -14, -15  n: .word 8  .text  main:  la a0, A # a0 = address(A[0])  lw a1, n # size of array  jal insertion\_sort  la a0, A # a0 = address(A[0])  lw a1, n # reload size of array  exit:  li a7, 10  ecall  # --------------------------------------------------------------  # Procedure insertion\_sort  # register usage:  # a0: pointer to base of array A  # a1: number of elements in array (n)  # t0: loop index i  # t1: loop index j  # t2: key element to be inserted  # t3: address of A[j]  # t4: key value  # t5: temporary storage for A[j]  # --------------------------------------------------------------  insertion\_sort:  addi t0, zero, 1 # i starts from 1 (second element)  outer\_loop:  bge t0, a1, end\_sort # if i >= n, end sorting  slli t1, t0, 2 # t1 = i \* 4 (byte offset for A[i])  add t3, a0, t1 # address of A[i]  lw t4, 0(t3) # key = A[i]  addi t1, t0, -1 # j = i - 1  inner\_loop:  blt t1, zero, insert\_key # if j < 0, insert the key  slli t5, t1, 2 # t5 = j \* 4 (byte offset for A[j])  add t6, a0, t5 # address of A[j]    lw t2, 0(t6) # load A[j] into t2 for comparison | bgt t2, t4, shift # if A[j] > key, shift A[j] right  j insert\_key  shift:  sw t2, 4(t6) # A[j+1] = A[j]  addi t1, t1, -1 # j--  j inner\_loop  insert\_key:  sw t4, 0(t6) # A[j+1] = key (insert the key)  addi t0, t0, 1 # i++  j outer\_loop  end\_sort:  jr ra # return from insertion\_sort |