Nguyễn Đình Dương – 20225966

Assignment 1

|  |
| --- |
| .data  A: .word 1, -5, 3, 7, -2, 8, -6 # Khởi tạo mảng A với các phần tử số nguyên  .text  main:  la a0, A # Tải địa chỉ cơ sở của mảng A vào thanh ghi a0  li a1, 8 # Gán giá trị 8 vào a1 (số phần tử trong mảng, nhưng thực tế mảng chỉ có 7 phần tử)  j mspfx # Nhảy đến thủ tục mspfx để tính tổng tiền tố lớn nhất  continue:  exit:  li a7, 10 # Gán giá trị 10 vào a7 (mã cho system call kết thúc chương trình)  ecall # Gọi system call để kết thúc chương trình  end\_of\_main:  # -----------------------------------------------------------------  # Thủ tục mspfx  # @brief Tìm tổng tiền tố lớn nhất trong danh sách các số nguyên  # @param[in] a0 Địa chỉ cơ sở của danh sách (A) cần xử lý  # @param[in] a1 Số phần tử trong danh sách (A)  # @param[out] s0 Độ dài của mảng con (sub-array) trong A mà tổng lớn nhất đạt được  # @param[out] s1 Tổng lớn nhất của một mảng con xác định  # -----------------------------------------------------------------  mspfx:  li s0, 0 # Khởi tạo độ dài của tiền tố lớn nhất trong s0 là 0  li s1, 0x80000000 # Khởi tạo tổng tiền tố lớn nhất trong s1 là số nguyên nhỏ nhất (giá trị nhỏ nhất kiểu int)  li t0, 0 # Khởi tạo chỉ số i cho vòng lặp trong t0 là 0  li t1, 0 # Khởi tạo tổng chạy (running sum) trong t1 là 0  loop:  add t2, t0, t0 # Đưa 2\*i vào t2 (tính 2\*i)  add t2, t2, t2 # Đưa 4\*i vào t2 (tính 4\*i)  add t3, t2, a0 # Đưa địa chỉ của A[i] vào t3 (tính 4\*i + A)  lw t4, 0(t3) # Tải giá trị A[i] từ bộ nhớ vào t4  add t1, t1, t4 # Cộng giá trị A[i] vào tổng chạy (running sum) trong t1  blt s1, t1, mdfy # Nếu tổng chạy (t1) lớn hơn tổng lớn nhất hiện tại (s1), nhảy đến mdfy để cập nhật kết quả  j next # Nhảy đến bước tiếp theo nếu không cần cập nhật  mdfy:  addi s0, t0, 1 # Cập nhật độ dài mới của tiền tố lớn nhất (i + 1)  addi s1, t1, 0 # Cập nhật tổng tiền tố lớn nhất là tổng chạy hiện tại  next:  addi t0, t0, 1 # Tăng chỉ số i lên 1  blt t0, a1, loop # Nếu i < n (số phần tử trong mảng), lặp lại  done:  j continue # Kết thúc thủ tục, nhảy về continue  mspfx\_end: |

Output:

Kết quả sau khi chạy vòng loop đầu tiên:

A table with numbers and a green line

Description automatically generatedKhởi tạo các giá trị đầu tiên, sau khi chạy vòng lặp đầu tiên thì nhảy vào branch mdfy tiến hành cập nhật giá trị s0,s1.

Kết quả sau khi chạy hết chương trình:A screenshot of a table

Description automatically generated

* **t0 (Number 5)**: 0x00000008 – Chỉ số hiện tại đã đạt tới phần tử thứ 8, cho thấy vòng lặp đã kết thúc.
* **t1 (Number 6)**: 0x00000006 – Tổng tiền tố hiện tại (running sum) cuối cùng là 6.
* **t2 (Number 7)**: 0x0000001c – Giá trị tạm thời sử dụng để tính toán địa chỉ mảng, không quan trọng sau khi chương trình kết thúc.
* **s0 (Number 8)**: 0x00000004 – Độ dài của tiền tố có tổng lớn nhất là 4 phần tử.
* **s1 (Number 9)**: 0x0000000c – Tổng tiền tố lớn nhất là 12.
* **a0 (Number 10)**: 0x10101000 – Địa chỉ cơ sở của mảng A.
* **a1 (Number 11)**: 0x00000008 – Số phần tử của mảng, ban đầu là 8.
* **pc (Program Counter)**: 0x00400018 – Địa chỉ tiếp theo của lệnh sẽ thực thi.

Tóm lại:

* **Tổng tiền tố lớn nhất là 12**.
* **Độ dài tiền tố có tổng lớn nhất là 4 phần tử**.

**Assignment 2**

|  |
| --- |
| .data  A: .word 5,6,1,4,3,2  Aend: .word  newline: .string "\n"  space: .string " "  .text  main:  la a2, A # a2 = address(A[0])  la a3, Aend  addi a3, a3, -4 # a3 = address(A[n-1])  mv a6, a3 # create a6 always = last elements  j sort # sort  after\_sort:  jal print\_array  li a7, 10  ecall  end\_main:  # --------------------------------------------------------------  # Procedure sort (ascending selection sort using pointer)  # register usage in sort program  # a2 pointer to the first element in unsorted part  # a3 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 a2, a3, done # single element list is sorted  j max # call the max procedure  after\_max:  lw t0, 0(a3) # load last element into $t0  sw t0, 0(s0) # copy last element to max location  sw s1, 0(a3) # copy max value to last element  addi a3, a3, -4 # decrement pointer to last element  jal print\_array  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  # a2 pointer to first element  # a3 pointer to last element  # ---------------------------------------------------------------------  max:  addi s0, a2, 0 # init max pointer to first element  lw s1, 0(s0) # init max value to first value  addi t0, a2, 0 # init next pointer to first  loop:  beq t0, a3, 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    print\_array:  mv a4,a2 # saved the address value of a2  mv a5,a6 # saved the address value of a6  print\_loop:  lw a0,0(a2)  li a7, 1 # print integer  ecall    la a0, space # load address of space string  li a7, 4 # print string  ecall    addi a2, a2, 4 # move to next element  ble a2, a6, print\_loop # if not past last element, continue loop    la a0, newline # print newline  li a7, 4  ecall  add a2, a4, zero # restore a2  add a6, a5, zero # restore a6  li a0, 0 # restore a0    ret # return to caller |

Output:

A screenshot of a computer

Description automatically generated

Kết quả chạy đúng như lý thuyết, đưa số lớn nhất về cuối

* + Đúng với lý thuyết

**Assignment 3**

Ý tưởng chính của thuật toán sắp xếp nổi bọt (Bubble Sort) như sau:

1. So sánh từng cặp phần tử liền kề trong mảng.
2. Nếu phần tử đứng trước lớn hơn phần tử đứng sau, hoán đổi vị trí của chúng.
3. Lặp lại quá trình này cho toàn bộ mảng, từ đầu đến cuối.
4. Sau mỗi lần lặp, phần tử lớn nhất trong số các phần tử chưa được sắp xếp sẽ "nổi" lên cuối mảng.
5. Lặp lại các bước 1-4 cho đến khi không còn cặp nào cần hoán đổi nữa.

Code:

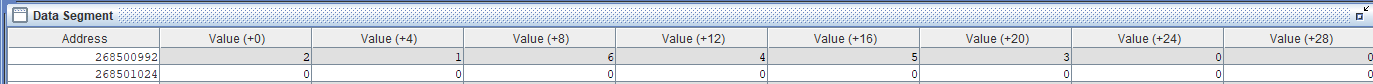
|  |
| --- |
| .data  A: .word 2, 1, 6, 4, 5, 3 # New array  Aend: .word # End array  newline: .string "\n" # String for new line  space: .string " " # String for space  .text  .globl main  main:  la a2, A # Load address of the beginning of array A into a2  la a3, Aend # Load address of the end of array A into a3  mv a6, a3 # Copy the end address of A into a6  addi a6, a6, -1 # Decrement a6 to point to the last element of A  li s0, 0 # Initialize s0 to 0, used as the element count  li s1, -1 # Initialize s1 to -1, used as the loop index i  DemPhanTu: # Count elements in the array  beq a3, a2, Size # If a3 == a2, reached the end of the array, jump to Size  addi a3, a3, -4 # Decrement a3 by 4 bytes (each element is 4 bytes)  addi s0, s0, 1 # Increment the element count (count)  j DemPhanTu # Repeat the loop  Size: # Calculate the size of the array  addi t0, s0, -1 # t0 = number of elements in array - 1  loop1: # Outer loop for sorting  addi s1, s1, 1 # Increment i  li s2, 0 # Initialize s2 to 0, used as the inner loop index j  beq s1, t0, Exit # If i == size - 1, jump to Exit  loop2: # Inner loop for comparison  sub t2, t0, s1 # t2 = (size - 1) - i  beq s2, t2, loop1 # If j == (size - 1) - i, return to loop1  if\_swap: # Check if a swap is needed  slli t3, s2, 2 # t3 = j \* 4 (calculate offset for address A[j])  add s3, a2, t3 # s3 = address of A[j]  lw t4, 0(s3) # Load value of A[j] into t4  addi s3, s3, 4 # s3 = address of A[j+1]  lw t5, 0(s3) # Load value of A[j+1] into t5  blt t5, t4, swap # If A[j+1] < A[j], jump to swap  addi s2, s2, 1 # Increment j  j loop2 # Repeat loop2  swap: # Swap A[j] and A[j+1]  sw t4, 0(s3) # A[j+1] = A[j]  addi s3, s3, -4 # s3 = address of A[j] (calculate back to A[j])  sw t5, 0(s3) # A[j] = A[j+1]  addi s2, s2, 1 # Increment j  jal print\_array # Call print\_array to display the current state of the array  j loop2 # Repeat loop2  print\_array: # Print the array  mv a4, a2 # Save the address value of a2 (start of the array)  mv a5, a6 # Save the address value of a6 (end of the array)  print\_loop:  lw a0, 0(a2) # Load the current element into a0  li a7, 1 # Syscall code for printing an integer  ecall # Make the syscall    la a0, space # Load address of space string  li a7, 4 # Syscall code for printing a string  ecall # Make the syscall    addi a2, a2, 4 # Move to the next element  ble a2, a6, print\_loop # If not past the last element, continue loop    la a0, newline # Load address of newline string  li a7, 4 # Syscall code for printing a string  ecall # Make the syscall    add a2, a4, zero # Restore a2 to its original value  add a6, a5, zero # Restore a6 to its original value  li a0, 0 # Restore a0  ret # Return to the caller  Exit: # Exit point of the program  li a7, 10 # Load syscall code to exit the program  ecall # Make the syscall |

Output:

A screenshot of a computer

Description automatically generated

Khi mới run chương trình, ta có thể quan sát thứ tự mảng như sau:



Qua từng bước :

|  |  |
| --- | --- |
| Loop 1: |  |
| Loop 2: |  |
| Loop 3: |  |
| Loop 4: |  |
| Loop 5: |  |
| Loop 6: |  |

Vậy là sau các vòng lặp thì mảng đã được sắp xếp về đúng vị trí.

* + Kết quả đúng với lý thuyết

**Assignment 4**

Ý tưởng chính của thuật toán sắp xếp chèn như sau:

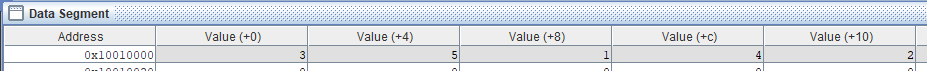
1. Chia mảng thành hai phần: phần đã sắp xếp và phần chưa sắp xếp.
2. Ban đầu, phần đã sắp xếp chỉ chứa phần tử đầu tiên của mảng.
3. Lặp qua từng phần tử trong phần chưa sắp xếp:
   * Lấy phần tử hiện tại (gọi là "key").
   * So sánh "key" với các phần tử trong phần đã sắp xếp, từ phải qua trái.
   * Di chuyển các phần tử lớn hơn "key" sang phải một vị trí.
   * Chèn "key" vào vị trí thích hợp trong phần đã sắp xếp.
4. Lặp lại bước 3 cho đến khi tất cả các phần tử đều đã được xét và chèn vào đúng vị trí.

Code:

Output:

|  |
| --- |
| .data  array: .word 3,5,1,4,2 # Array to be sorted  arrayEnd: .word # End of the array  newline: .string "\n" # String for new line  space: .string " " # String for space  .text  .globl main  main:  la a2, array # Load address of the beginning of the array into a2  la a6, arrayEnd # Load address of the end of the array into a6  addi a3, a3, 5 # Set the number of elements (n) to 5  addi a6, a6, -4 # Adjust a6 to point to the last element of the array  jal ra, insertion\_sort # Call insertion\_sort function    # End the program  li a7, 10 # Load syscall code for program exit  ecall # Make the syscall  insertion\_sort:  li t0, 1 # i = 1 (starting from the second element)  outer\_loop:  bge t0, a3, done # If i >= n, exit the loop  slli t1, t0, 2 # t1 = i \* 4 (offset for the i-th element)  add t1, a2, t1 # Get the address of a[i]  lw t2, 0(t1) # key = a[i]  addi t3, t0, -1 # j = i - 1    inner\_loop:  bltz t3, insert # If j < 0, insert key  slli t4, t3, 2 # t4 = j \* 4 (offset for the j-th element)  add t4, a2, t4 # Get the address of a[j]  lw t5, 0(t4) # Load value of a[j]  ble t5, t2, insert # If a[j] <= key, insert key    sw t5, 4(t4) # a[j+1] = a[j] (shift element to the right)  addi t3, t3, -1 # j-- (decrement j)  j inner\_loop # Repeat inner loop    insert:  slli t4, t3, 2 # Calculate offset for j  add t4, a2, t4 # Get address of a[j]  sw t2, 4(t4) # a[j+1] = key (insert the key at the correct position)  jal print\_array # Call print\_array to display the current state of the array  addi t0, t0, 1 # i++ (increment i)  j outer\_loop # Repeat outer loop  print\_array: # Print the array  mv a4, a2 # Save the address value of a2 (start of the array)  mv a5, a6 # Save the address value of a6 (end of the array)  print\_loop:  lw a0, 0(a2) # Load the current element into a0  li a7, 1 # Syscall code for printing an integer  ecall # Make the syscall    la a0, space # Load address of space string  li a7, 4 # Syscall code for printing a string  ecall # Make the syscall    addi a2, a2, 4 # Move to the next element  ble a2, a6, print\_loop # If not past the last element, continue loop    la a0, newline # Load address of newline string  li a7, 4 # Syscall code for printing a string  ecall # Make the syscall    mv a2, a4 # Restore a2 to its original value  mv a6, a5 # Restore a6 to its original value  li a0, 0 # Restore a0  ret # Return to the caller  done:  li a7, 10 # Load syscall code to exit the program  ecall # Make the syscall |

Kết quả khi mới khởi tạo mảng:



|  |  |
| --- | --- |
| Loop 1: |  |
| Loop 2: |  |
| Loop 3: |  |

A screenshot of a computer

Description automatically generated

* + Kết quả đúng với lý thuyết