**REPORT WEEK 6**

Họ và tên: Nguyễn Văn Hưng

MSSV: 20225634

Lớp: Thực hành kiến trúc máy tính – Mã lớp: 147799

1. **Assignment 1:**
2. Đề bài: Create a new project to implement procedure in Home Assignment 1. Add code the main program and initialize data for the integer list. Compile and upload to simulator. Run this program step by step, observe the process of explore each element of the integer list using indexing method.
3. Code:

.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-sum prefix in a list of integers

# @param[in] a0 the base address of this list(A) need to be processed

# @param[in] a1 the number of elements in list(A)

# @param[out] v0 the length of sub-array of A in which max sum reachs.

# @param[out] v1 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: addi $v0,$zero,0 #initialize length in $v0 to 0

addi $v1,$zero,0 #initialize max sum in $v1to 0

addi $t0,$zero,0 #initialize index i in $t0 to 0

addi $t1,$zero,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

slt $t5,$v1,$t1 #set $t5 to 1 if max sum < new sum

bne $t5,$zero,mdfy #if 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

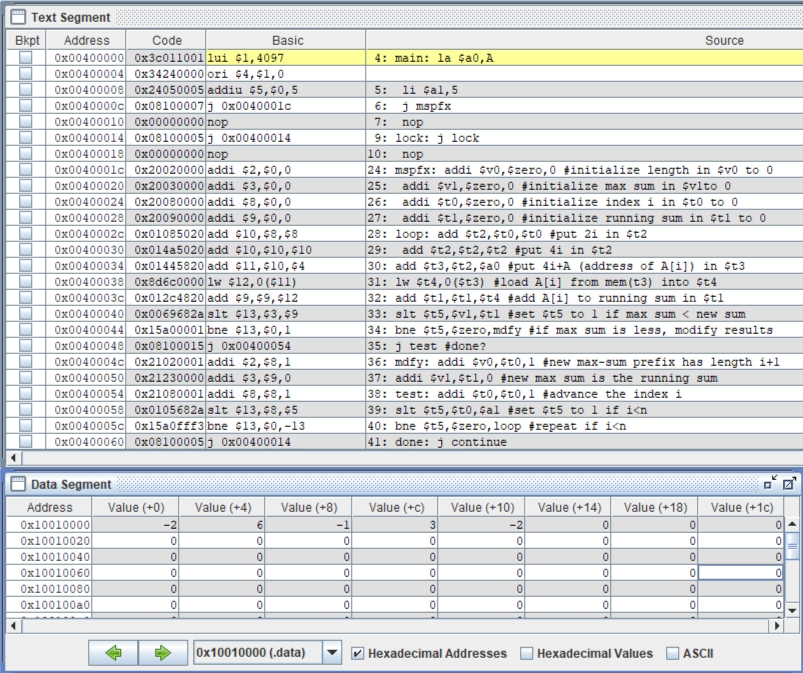
test: addi $t0,$t0,1 #advance the index i

slt $t5,$t0,$a1 #set $t5 to 1 if i<n

bne $t5,$zero,loop #repeat if i<n

done: j continue

mspfx\_end:

1. Text segment and data segment: 
2. Giải thích

Đoạn mã này thực hiện việc tìm chuỗi con có tổng lớn nhất (maximum-sum prefix) trong một danh sách các số nguyên. Dưới đây là phân tích chi tiết của nó:

**.data**: Phần dữ liệu.

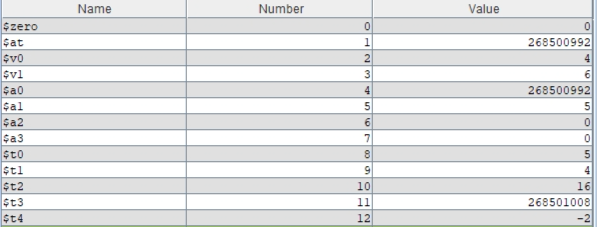
* + **A: .word -2, 6, -1, 3, -2**: Khai báo một mảng các số nguyên.

**.text**: Phần mã chương trình.

* + **main:**: Nhãn định nghĩa hàm chính **main**.
    - **la $a0,A**: Đưa địa chỉ của mảng **A** vào thanh ghi $a0.
    - **li $a1,5**: Gán giá trị 5 vào thanh ghi $a1, đây là số lượng phần tử trong mảng.
    - **j mspfx**: Nhảy đến nhãn **mspfx** để thực hiện quá trình tìm chuỗi con có tổng lớn nhất.
    - **nop**: Lệnh không làm gì, dùng để đảm bảo việc nhảy đến **mspfx** được thực hiện một cách chính xác.
  + **continue:**: Nhãn định nghĩa điểm tiếp tục chương trình.
  + **lock:**: Nhãn định nghĩa vùng khóa.
    - **j lock**: Lệnh vô hạn lặp, dùng để tạo ra một vùng khóa vô hạn.
    - **nop**: Lệnh không làm gì, dùng để đảm bảo vòng lặp vô hạn.
  + **end\_of\_main:**: Nhãn kết thúc chương trình chính.
  + **mspfx:**: Nhãn định nghĩa hàm **mspfx**.
    - **addi $v0,$zero,0**: Khởi tạo độ dài chuỗi con có tổng lớn nhất ($v0) bằng 0.
    - **addi $v1,$zero,0**: Khởi tạo tổng lớn nhất của chuỗi con ($v1) bằng 0.
    - **addi $t0,$zero,0**: Khởi tạo biến đếm $t0 (index i) bằng 0.
    - **addi $t1,$zero,0**: Khởi tạo tổng tạm thời ($t1) bằng 0.
    - **loop:**: Nhãn bắt đầu vòng lặp.
      * **add $t2,$t0,$t0**: Nhân biến đếm i với 2 và lưu kết quả vào $t2.
      * **add $t2,$t2,$t2**: Nhân $t2 với 2 và lưu kết quả vào $t2 (tương đương với việc nhân biến đếm i với 4).
      * **add $t3,$t2,$a0**: Tính địa chỉ của phần tử thứ i trong mảng A và lưu vào $t3.
      * **lw $t4,0($t3)**: Load giá trị của phần tử thứ i trong mảng A vào $t4.
      * **add $t1,$t1,$t4**: Cộng giá trị của phần tử thứ i vào tổng tạm thời.
      * **slt $t5,$v1,$t1**: So sánh tổng tạm thời với tổng lớn nhất hiện tại, nếu tổng tạm thời lớn hơn, thực hiện điều chỉnh.
      * **bne $t5,$zero,mdfy**: Nếu tổng tạm thời lớn hơn tổng lớn nhất hiện tại, nhảy đến nhãn **mdfy** để điều chỉnh kết quả.
      * **j test**: Kiểm tra xem đã hoàn thành việc duyệt qua tất cả các phần tử trong mảng chưa.
    - **mdfy:**: Nhãn thực hiện việc điều chỉnh kết quả.
      * **addi $v0,$t0,1**: Gán giá trị i+1 vào thanh ghi $v0, đây là độ dài của chuỗi con có tổng lớn nhất.
      * **addi $v1,$t1,0**: Gán giá trị tổng tạm thời vào thanh ghi $v1, đây là tổng lớn nhất của chuỗi con.
    - **test:**: Kiểm tra xem đã duyệt qua tất cả các phần tử trong mảng chưa.
      * **addi $t0,$t0,1**: Tăng biến đếm i lên 1.
      * **slt $t5,$t0,$a1**: Kiểm tra xem đã duyệt qua tất cả các phần tử trong mảng chưa.
      * **bne $t5,$zero,loop**: Nếu chưa duyệt hết, nhảy đến nhãn **loop** để tiếp tục duyệt.
    - **done:**: Nhãn thực hiện việc kết thúc quá trình tìm chuỗi con có tổng lớn nhất.
    - **j continue**: Nhảy đến nhãn **continue** để tiếp tục thực hiện chương trình chính.
    - **mspfx\_end:**: Nhãn kết thúc hàm **mspfx**.

1. Kết quả

Dưới đây là giá trị của các thanh ghi sau từng câu lệnh:

* Sau lệnh **addi $v0,$zero,0**: $v0 = 0, $v1 = 0, $t0 = 0, $t1 = 0.
* Sau lệnh **addi $v1,$zero,0**: $v0 = 0, $v1 = 0, $t0 = 0, $t1 = 0.
* Sau lệnh **addi $t0,$zero,0**: $v0 = 0, $v1 = 0, $t0 = 0, $t1 = 0.
* Sau lệnh **addi $t1,$zero,0**: $v0 = 0, $v1 = 0, $t0 = 0, $t1 = 0.
* Sau lệnh **loop:**:
  + Trong vòng lặp đầu tiên ($t0 = 0):
    - $t2 = 0, $t3 = Địa chỉ của **A[0]**, $t4 = -2, $t1 = -2.
    - $v0 = 0, $v1 = 0, $t0 = 1, $t1 = -2.
  + Trong vòng lặp thứ hai ($t0 = 1):
    - $t2 = 4, $t3 = Địa chỉ của **A[1]**, $t4 = 6, $t1 = 4.
    - $v0 = 0, $v1 = 4, $t0 = 2, $t1 = 4.
  + Trong vòng lặp thứ ba ($t0 = 2):
    - $t2 = 8, $t3 = Địa chỉ của **A[2]**, $t4 = -1, $t1 = 3.
    - $v0 = 0, $v1 = 4, $t0 = 3, $t1 = 3.
  + Trong vòng lặp thứ tư ($t0 = 3):
    - $t2 = 12, $t3 = Địa chỉ của **A[3]**, $t4 = 3, $t1 = 6.
    - $v0 = 0, $v1 = 6, $t0 = 4, $t1 = 6.
  + Trong vòng lặp thứ năm ($t0 = 4):
    - $t2 = 16, $t3 = Địa chỉ của **A[4]**, $t4 = -2, $t1 = 4.
    - $v0 = 0, $v1 = 6, $t0 = 5, $t1 = 4.
* Sau lệnh **slt $t5,$v1,$t1**:
  + $v0 = 0, $v1 = 6, $t0 = 5, $t1 = 4, $t5 = 1.
* Sau lệnh **bne $t5,$zero,mdfy**:
  + $v0 = 0, $v1 = 6, $t0 = 5, $t1 = 4.
* Sau lệnh **j test**:
  + $v0 = 0, $v1 = 6, $t0 = 5, $t1 = 4.
* Sau lệnh **mdfy:**:
  + $v0 = 5, $v1 = 6, $t0 = 5, $t1 = 4.
* Sau lệnh **addi $v0,$t0,1**:
* $v0 = 6, $v1 = 6, $t0 = 5, $t1 = 4.
* Sau lệnh **addi $v1,$t1,0**:
* $v0 = 6, $v1 = 4, $t0 = 5, $t1 = 4.
* Sau lệnh **test:**:
* $v0 = 6, $v1 = 4, $t0 = 6, $t1 = 4.
* Sau lệnh **slt $t5,$t0,$a1**:
* $v0 = 6, $v1 = 4, $t0 = 6, $t1 = 4, $t5 = 0.
* Sau lệnh **bne $t5,$zero,loop**:
* $v0 = 6, $v1 = 4, $t0 = 6, $t1 = 4.
* Sau lệnh **done:**:
* $v0 = 6, $v1 = 4, $t0 = 6, $t1 = 4.
* Sau lệnh **j continue**:
* $v0 = 6, $v1 = 4, $t0 = 6, $t1 = 4.
* Sau lệnh **mspfx\_end:**:
* $v0 = 6, $v1 = 4, $t0 = 6, $t1 = 4.
  + - Top of Form
* Kết quả cuối cùng ta thu được tổng là 6
* 

1. **Assignment 2:**
2. Đề bài: Create a new project to implement procedure in Home Assignment 2. Add code the main program and initialize data for the integer list. Compile and upload to simulator. Run this program step by step, observe the process of explore each element of the integer list using pointer updating method. Write a procedure to print array after each round.
3. Code:

.data  
n: .word 10  
a: .word 10, 60, 40, 70, 20, 30, 90, 100, 0, 80, 50  
space: .asciiz " "  
newline: .asciiz "\n"

.text  
.globl main

main:  
    # Load n and the base address of array a  
    lw $t0, n           # Load n into $t0  
    la $t1, a           # Load the base address of array a into $t1

    # Outer loop: i = 0 to n-2  
    li $t2, 0           # i = 0  
outer\_loop:  
    # Load i into $t3  
    move $t3, $t2       # $t3 = i

    # Inner loop: j = i+1 to n-1  
inner\_loop:  
    # Calculate the addresses of a[i] and a[j]  
    mul $t4, $t2, 4     # Calculate the offset (i\*4)  
    add $t5, $t1, $t4   # Calculate the address of a[i]  
    addi $t6, $t3, 1    # j = i + 1  
    mul $t6, $t6, 4     # Calculate the offset (j\*4)  
    add $t7, $t1, $t6   # Calculate the address of a[j]

    # Load a[i] and a[j]  
    lw $t8, 0($t5)      # Load a[i] into $t8  
    lw $t9, 0($t7)      # Load a[j] into $t9

    # Compare a[i] and a[j]  
    ble $t8, $t9, not\_swap

    # Swap a[i] and a[j]  
    sw $t9, 0($t5)      # Store a[j] into a[i]  
    sw $t8, 0($t7)      # Store a[i] into a[j]

not\_swap:  
    # Increment j  
    addi $t3, $t3, 1    # Increment j

    # Check if j < n  
    blt $t3, $t0, inner\_loop

    # Increment i  
    addi $t2, $t2, 1    # Increment i

    # Check if i < n-1  
    blt $t2, $t0, outer\_loop

    # Print sorted array  
    li $v0, 4            # Print string syscall  
    la $a0, newline  
    syscall

    li $v0, 4            # Print string syscall  
    la $a0, newline  
    syscall

    li $v0, 4            # Print string syscall  
    la $a0, newline  
    syscall

    li $v0, 4            # Print string syscall  
    la $a0, newline  
    syscall

    li $v0, 4            # Print string syscall  
    la $a0, newline  
    syscall

    li $v0, 10           # Exit syscall  
    syscall

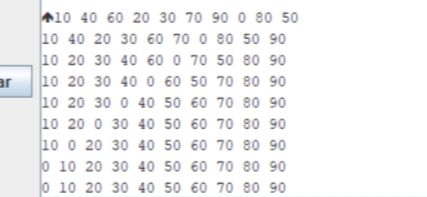
1. Text segment and data segment: A screenshot of a computer

   Description automatically generated
2. Giải thích:

Code MIPS assembly trên là một chương trình sắp xếp mảng theo thứ tự tăng dần bằng cách sử dụng thuật toán "selection sort" với việc sử dụng con trỏ.

* **main**:
  + Khởi tạo con trỏ **$a0** trỏ đến phần tử đầu tiên của mảng **A** và **$a1** trỏ đến phần tử cuối cùng của mảng **A**.
  + Gọi thủ tục **sort** để sắp xếp mảng.
  + Sau khi sắp xếp xong, thoát chương trình.
* **sort**:
  + Kiểm tra xem nếu chỉ còn một phần tử trong danh sách thì danh sách đã được sắp xếp và quay trở lại **after\_sort**.
  + Gọi thủ tục **max** để tìm phần tử lớn nhất trong mảng chưa sắp xếp.
  + Sau khi tìm được phần tử lớn nhất, đổi chỗ phần tử cuối cùng chưa sắp xếp với phần tử lớn nhất.
  + Giảm con trỏ của phần tử cuối cùng chưa sắp xếp và lặp lại quá trình cho đến khi mảng được sắp xếp hoàn toàn.
* **max**:
  + Khởi tạo con trỏ **$v0** trỏ đến phần tử đầu tiên của mảng.
  + Khởi tạo giá trị **$v1** bằng giá trị của phần tử đầu tiên của mảng.
  + Tiến hành lặp qua từng phần tử trong mảng, tìm ra phần tử lớn nhất.
  + Khi tìm thấy phần tử lớn nhất, trả về con trỏ và giá trị của phần tử đó.

1. Kết quả:



1. **Assignment 3:**
2. Đề bài: Write a procedure to implement bubble sort algorithm.
3. Code:

.data  
n: .word 10  
a: .word 10, 60, 40, 70, 20, 30, 90, 100, 0, 80, 50  
space: .asciiz " "  
newline: .asciiz "\n"

.text  
.globl main

main:  
    # Load n and the base address of array a  
    lw $t0, n           # Load n into $t0  
    la $t1, a           # Load the base address of array a into $t1

    # Outer loop: i = 0 to n-1  
    li $t2, 0           # i = 0  
outer\_loop:  
    li $t3, 0           # j = 0

    # Inner loop: j = 0 to n-i-1  
inner\_loop:  
    # Calculate the address of a[j] and a[j+1]  
    mul $t4, $t3, 4     # Calculate the offset (j\*4)  
    add $t5, $t1, $t4   # Calculate the address of a[j]  
    addi $t6, $t5, 4    # Calculate the address of a[j+1]

    # Load a[j] and a[j+1]  
    lw $t7, 0($t5)      # Load a[j] into $t7  
    lw $t8, 0($t6)      # Load a[j+1] into $t8

    # Compare a[j] and a[j+1]  
    ble $t7, $t8, not\_swap

    # Swap a[j] and a[j+1]  
    sw $t8, 0($t5)      # Store a[j+1] into a[j]  
    sw $t7, 0($t6)      # Store a[j] into a[j+1]

not\_swap:  
    # Increment j  
    addi $t3, $t3, 1    # Increment j

    # Check if j < n-i-1  
    blt $t3, $t0, inner\_loop

    # Print array after this iteration  
    li $t3, 0            # Reset j  
    print\_loop:  
        mul $t4, $t3, 4    # Calculate the offset (j\*4)  
        add $t5, $t1, $t4  # Calculate the address of a[j]

        lw $a0, 0($t5)     # Load current array element  
        li $v0, 1          # Print integer syscall  
        syscall

        li $v0, 4          # Print space syscall  
        la $a0, space  
        syscall

        addi $t3, $t3, 1   # Increment j  
        blt $t3, $t0, print\_loop  # Continue print loop if j < n

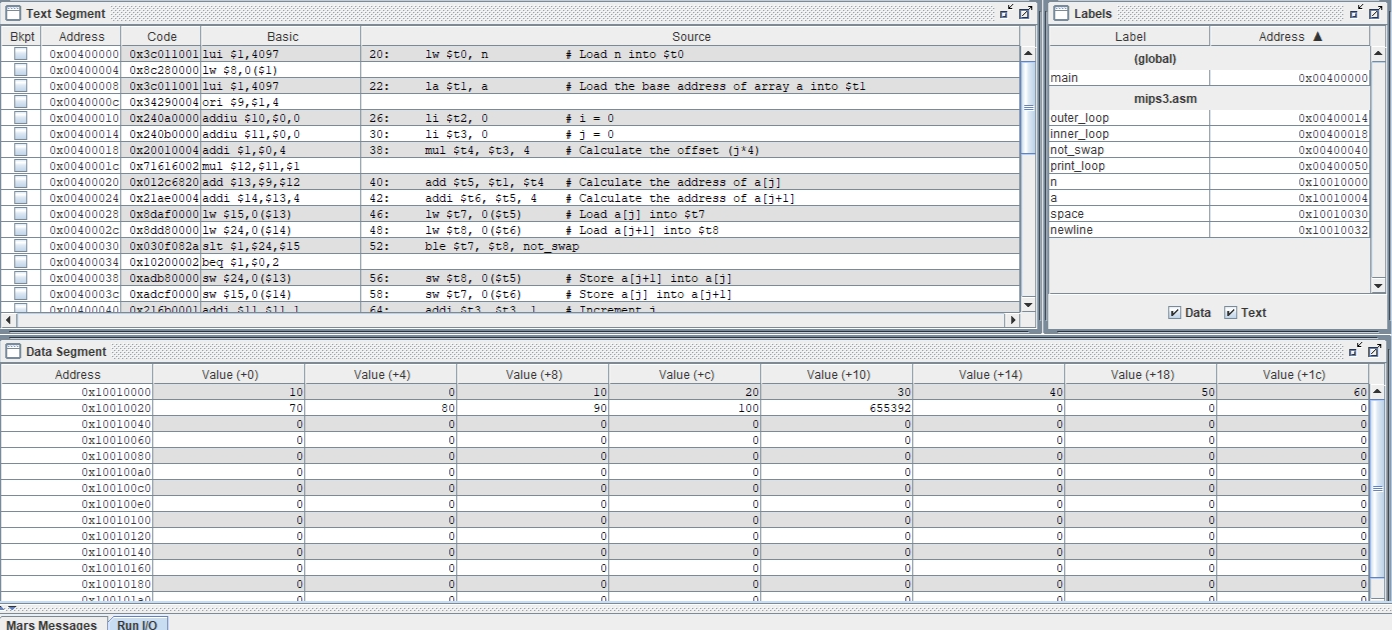
    # Print newline  
    li $v0, 4            # Print string syscall  
    la $a0, newline  
    syscall

    # Increment i  
    addi $t2, $t2, 1    # Increment i

    # Check if i < n-1  
    blt $t2, $t0, outer\_loop

    # Exit program  
    li $v0, 10           # Exit syscall  
    syscall

1. Text segment and data segment:

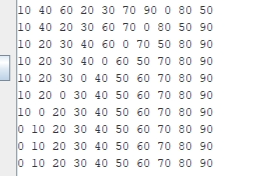


1. Giải thích

Đoạn code MIPS trên là một chương trình sử dụng thuật toán Bubble Sort để sắp xếp mảng **a**. Dưới đây là giải thích từng bước trong chương trình:

* + **.data** section: Định nghĩa các biến dữ liệu, bao gồm **n** (số phần tử trong mảng), mảng **a**, và các chuỗi **" "** và **"\n"** (dùng để in khoảng trắng và xuống dòng).
  + **.text** section:
    - **main**: Hàm main bắt đầu.
    - Load giá trị của **n** vào thanh ghi **$t0** và địa chỉ cơ sở của mảng **a** vào **$t1**.
  + Vòng lặp bên ngoài (**outer\_loop**): Duyệt qua mảng từ **i = 0** đến **n - 1**.
    - **i** được lưu trong **$t2**.
    - Ban đầu, **i = 0**.
  + Vòng lặp bên trong (**inner\_loop**): Duyệt qua mảng từ **j = 0** đến **n - i - 1**.
    - **j** được lưu trong **$t3**.
    - Ban đầu, **j = 0**.
  + So sánh hai phần tử liên tiếp **a[j]** và **a[j+1]**. Nếu **a[j] > a[j+1]**, thì hoán đổi chúng.
  + Sau khi hoàn thành vòng lặp bên trong, in mảng sau mỗi lần duyệt để theo dõi quá trình sắp xếp.
  + Vòng lặp bên ngoài tiếp tục lặp, và quá trình sắp xếp được lặp lại.
  + Cuối cùng, chương trình thoát.

1. Kết quả



1. **Assignment 4:**
2. Đề bài: Write a procedure to implement insertion sort algorithm
3. Code:

.data

n: .word 10

a: .word 10, 60, 40, 70, 20, 30, 90, 100, 0, 80, 50

space: .asciiz " "

newline: .asciiz "\n"

.text

.globl main

main:

# Load n and the base address of array a

lw $t0, n # Load n into $t0

la $t1, a # Load the base address of array a into $t1

# Outer loop: i = 1 to n-1

li $t2, 1 # i = 1

outer\_loop:

# Inner loop: j = i to 1

move $t3, $t2 # j = i

inner\_loop:

# Calculate the address of a[j] and a[j-1]

mul $t4, $t3, 4 # Calculate the offset (j\*4)

add $t5, $t1, $t4 # Calculate the address of a[j]

sub $t6, $t3, 1 # Calculate j - 1

mul $t6, $t6, 4 # Calculate the offset (j-1)\*4

add $t7, $t1, $t6 # Calculate the address of a[j-1]

# Load a[j] and a[j-1]

lw $t8, 0($t5) # Load a[j] into $t8

lw $t9, 0($t7) # Load a[j-1] into $t9

# Compare a[j-1] and a[j]

bge $t9, $t8, not\_swap

# Swap a[j-1] and a[j]

sw $t8, 0($t7) # Store a[j] into a[j-1]

sw $t9, 0($t5) # Store a[j-1] into a[j]

not\_swap:

# Decrement j

subi $t3, $t3, 1 # Decrement j

# Check if j > 0

bgtz $t3, inner\_loop

# Print array after this iteration

li $t3, 0 # Reset j

print\_loop:

mul $t4, $t3, 4 # Calculate the offset (j\*4)

add $t5, $t1, $t4 # Calculate the address of a[j]

lw $a0, 0($t5) # Load current array element

li $v0, 1 # Print integer syscall

syscall

li $v0, 4 # Print space syscall

la $a0, space

syscall

addi $t3, $t3, 1 # Increment j

blt $t3, $t0, print\_loop # Continue print loop if j < n

# Print newline

li $v0, 4 # Print string syscall

la $a0, newline

syscall

# Increment i

addi $t2, $t2, 1 # Increment i

# Check if i < n-1

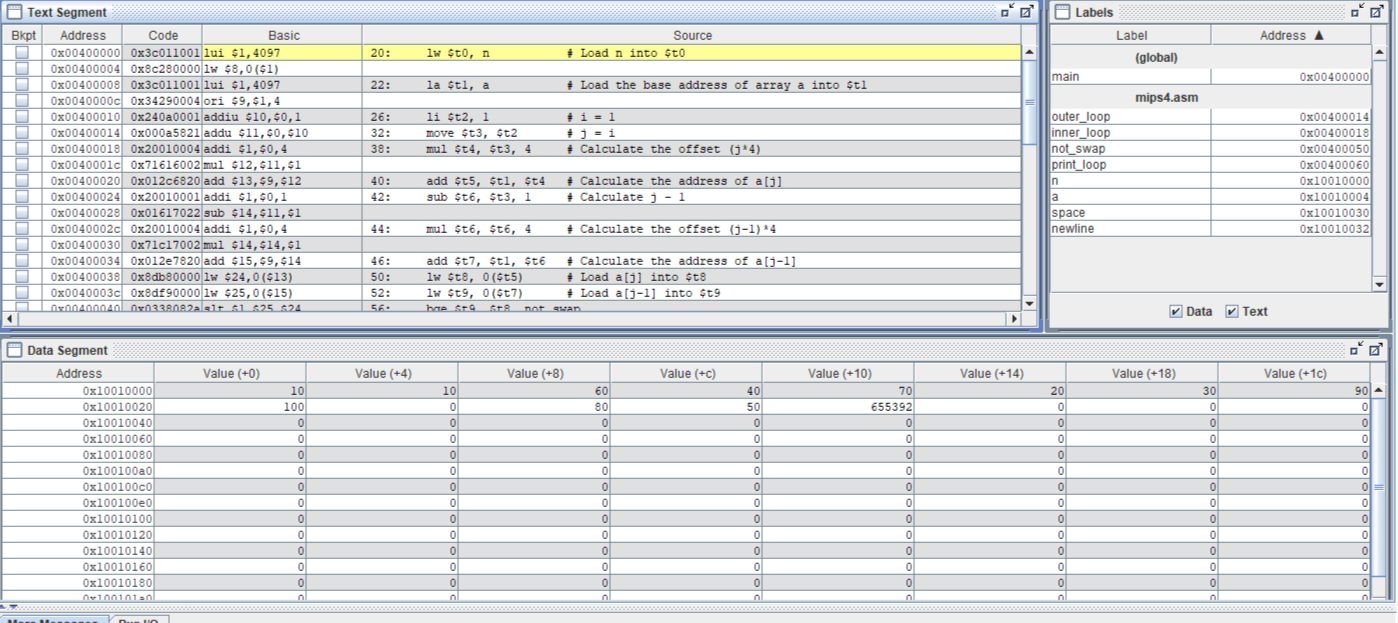
blt $t2, $t0, outer\_loop

# Exit program

li $v0, 10 # Exit syscall

syscall

1. Text segment and data segment:



1. Giải thích

* Chương trình trên là một chương trình MIPS assembly thực hiện thuật toán sắp xếp chèn (insertion sort) trên một mảng số nguyên đã cho và sau đó in ra mảng sau mỗi lần duyệt của thuật toán.
* Dưới đây là các phần chính của chương trình:
* **.data section**: Trong phần này, chúng ta định nghĩa các biến cần thiết, bao gồm kích thước của mảng n, mảng a chứa dữ liệu số nguyên cần sắp xếp, và các chuỗi để in ra (space và newline).
* **.text section**: Trong phần này, chúng ta viết mã lệnh thực hiện thuật toán sắp xếp chèn và in ra mảng.
* **main**: Phần main của chương trình bắt đầu bằng việc tải kích thước của mảng n và địa chỉ cơ sở của mảng a. Tiếp theo, chương trình thực hiện thuật toán sắp xếp chèn bằng cách lặp qua từng phần tử của mảng và chèn chúng vào vị trí phù hợp.
* **outer\_loop**: Trong vòng lặp này, chương trình duyệt qua từng phần tử của mảng, bắt đầu từ phần tử thứ hai (vì chúng ta giả sử rằng phần tử đầu tiên đã được sắp xếp).
* **inner\_loop**: Trong vòng lặp này, chương trình so sánh phần tử hiện tại với các phần tử trước đó và chèn phần tử hiện tại vào đúng vị trí trong dãy đã sắp xếp. Quá trình này được thực hiện bằng cách dịch chuyển các phần tử lớn hơn phần tử hiện tại sang phải cho đến khi tìm được vị trí thích hợp.
* **print\_loop**: Sau mỗi lần duyệt trong thuật toán sắp xếp chèn, chương trình in ra mảng hiện tại để kiểm tra tiến độ của thuật toán.
* **Exit program**: Cuối cùng, chương trình kết thúc bằng lệnh exit syscall để thoát khỏi chương trình.

1. Kết quả

A screenshot of a computer program

Description automatically generated