

BÁO CÁO THỰC HÀNH KIẾN TRÚC MÁY TÍNH TUẦN

6

Họ và tên: Hoàng Văn Thắng

MSSV: 20235828

Assignment 1

Tạo project thực hiện chương trình trong Home Assignment 1. Khởi tạo bộ giá trị mới cho mảng, dịch và nạp lên mô phỏng. Chạy chương trình từng bước một và quan sát sự thay đổi các thanh ghi để kiểm nghiệm chương trình hoạt động đúng với thuật toán.

Nhập chương trình:

```
.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:
# -----
# 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 reaches.
# @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                # 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:

```

Kết quả chạy:

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000001
t1	6	0x00000001
t2	7	0x00000000
s0	8	0x00000001
s1	9	0x00000001
a0	10	0x10010000
a1	11	0x00000008
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x10010000
t4	29	0x00000001
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400054

Khởi 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.

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000008
t1	6	0x00000006
t2	7	0x0000001c
s0	8	0x00000006
s1	9	0x0000000c
a0	10	0x10010000
a1	11	0x00000008
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x0000000a
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x1001001c
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400018

Kết quả sau khi chạy hết chương trình:

t0: 0x00000008 – Chỉ số hiện tại đã đạt tới phần tử thứ 8, cho thấy vòng lặp đã kết thúc.

t1: 0x00000006 – Tổng tiền tố hiện tại (running sum) cuối cùng là 6.

t2: 0x0000001c – Giá trị tạm thời sử dụng để tính toán địa chỉ mảng, không quan trọng sau khi chương trình kết thúc.

s0: 0x00000004 – Độ dài của tiền tố có tổng lớn nhất là 4 phần tử.

a0: 0x10101000 – Địa chỉ cơ sở của mảng A.

a1: 0x00000008 – Số phần tử của mảng, ban đầu là 8.

pc: 0x00400018 – Địa chỉ tiếp theo của lệnh sẽ thực thi.

Ta thấy:

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

Assignment 2

Tạo mới một project thực hiện chương trình trong Home Assignment 2. Khởi tạo bộ giá trị mới cho mảng, dịch và nạp lên mô phỏng. Chạy chương trình từng bước một và quan sát sự thay đổi các thanh ghi để kiểm nghiệm chương trình hoạt động đúng với thuật toán. Viết thêm chương trình con để in ra mảng sau mỗi lượt sắp xếp.

Nhập chương trình:

```
.data
A: .word 5,6,1,4,3,2
Aend: .word
newline: .string "\n"
space: .string " "
```

```

error_message: .asciz "Khong co phan tu nao de sap xep"
.text
main:
    la    a2, A        # a2 = address(A[0])
    la    a3, Aend
    beq   a2, a3, end # if array don't have element end of system
    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
# -----
error_print:
    li    a7, 4
    la    a0, error_message
    ecall
end:
    li    a7, 10
    ecall
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: find 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

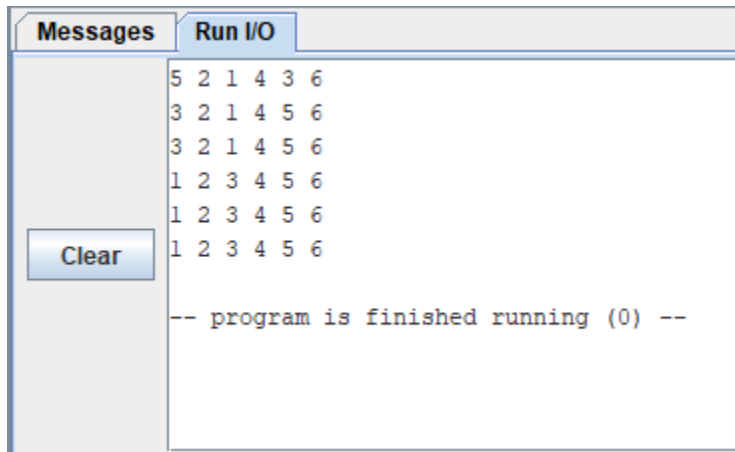
    la      a0, space    # load address of space string
    li      a7, 4        # print string
    ecall
    addi    a2, a2, 4     # move to next element
    ble     a2, a6, print_loop # if not past last element, continue loop

    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	

Kết quả chạy



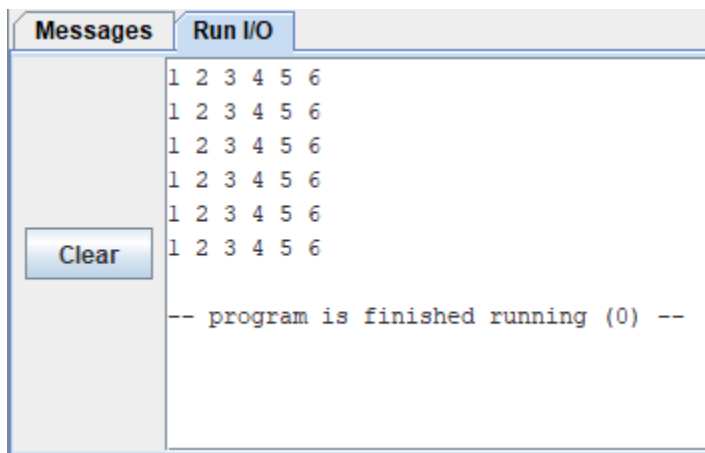
Kết quả chạy đúng như lý thuyết, đưa số lớn nhất về cuối

→ Đúng với lý thuyết

Chạy chương trình với các bộ test khác nhau

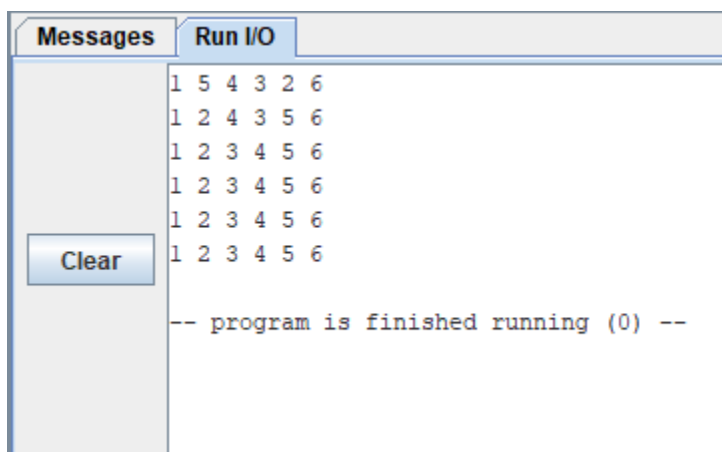
Case 1: Mảng đầu vào đã được sắp xếp theo thứ tự tăng dần (1, 2, 3, 4, 5, 6)

Kết quả chạy

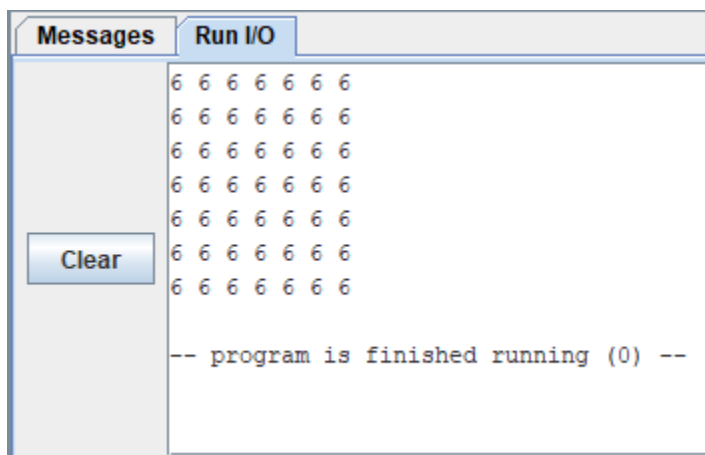


Selection Sort không dừng ngay khi gặp dữ liệu đầu vào là một dãy đã được sắp xếp

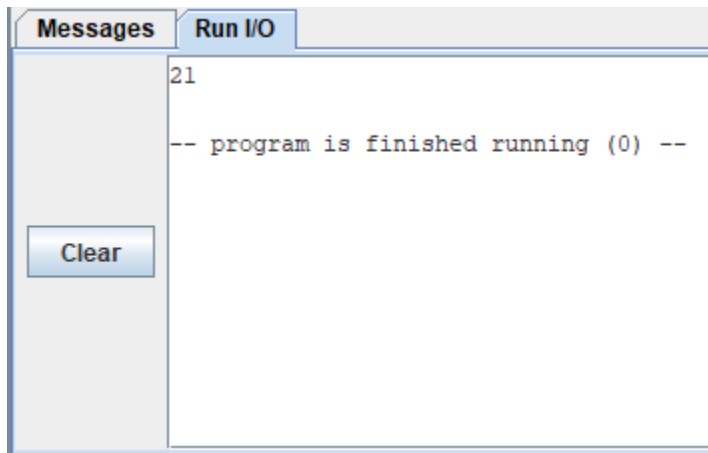
Case 2: Mảng đầu vào đã được sắp xếp theo thứ tự giảm dần (6, 5, 4, 3, 2, 1)



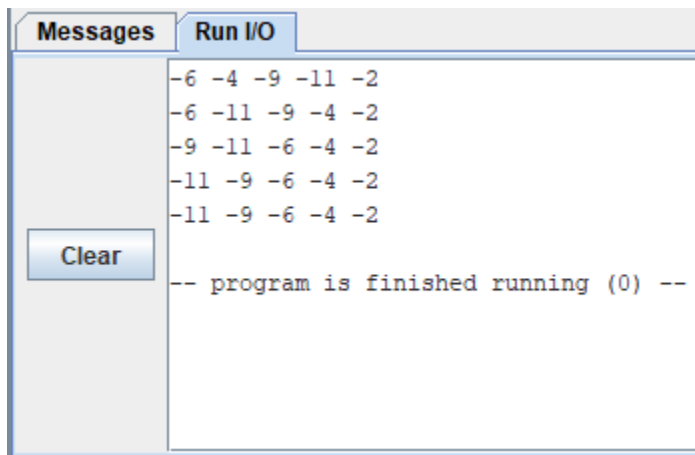
Case 3: Mảng có tất cả các phần tử bằng nhau (6, 6, 6, 6, 6, 6)



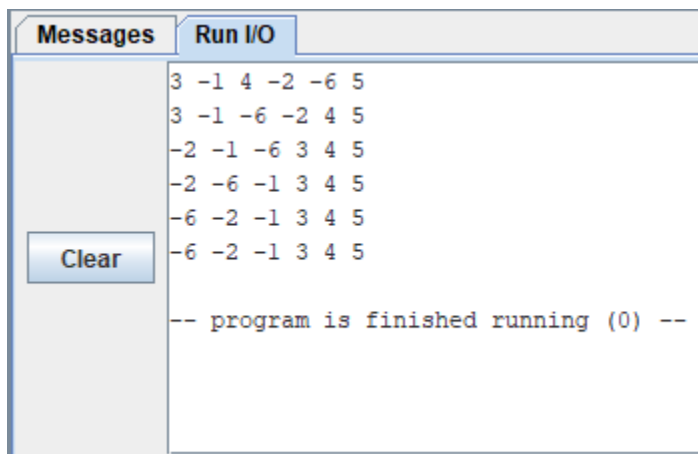
Case 4: Mảng chỉ có một phần tử duy nhất (21)



Case 5: Mảng chứa số nguyên âm (-6, -2, -9, -11, -4)



Case 6: Mảng chứa cả số nguyên dương và số nguyên âm (3, -1, 4, -2, 5, -6)



Case 7: Mảng có các phần tử trùng nhau (3, 4, 5, 3, 2, 5)

Messages	Run I/O
<div>Clear</div>	3 4 5 3 2 5
	3 4 2 3 5 5
	3 3 2 4 5 5
	3 2 3 4 5 5
	2 3 3 4 5 5
	2 3 3 4 5 5
	-- program is finished running (0) --

Case 8: Mảng không có phần tử nào

Messages	Run I/O
<div>Clear</div>	Khong co phan tu nao de sap xep
	-- program is finished running (0) --

Case 9: Mảng chứa phần tử 0 (0, 5, -2, 3, 1)

Messages	Run I/O
<div>Clear</div>	0 1 -2 3 5
	0 1 -2 3 5
	0 -2 1 3 5
	-2 0 1 3 5
	-2 0 1 3 5
	-- program is finished running (0) --

Assignment 3

Viết chương trình thực hiện thuật toán sắp xếp nổi bọt (bubble sort).

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.

Nhập chương trình

```
.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
error_message: .asciz "Khong co phan tu nao de sap xep"
.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
    beq a2, a3, error_print  # if array don't have element end of system
    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                # 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   # 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

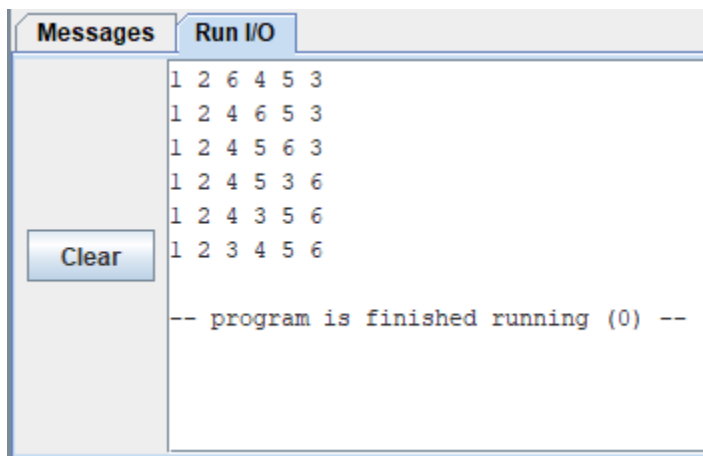
    ret                  # Return to the caller

Exit:                    # Exit point of the program
    li a7, 10            # Load syscall code to exit the program
    ecall                # Make the syscall

error_print:
    li a7, 4
    la a0, error_message
    ecall
    j Exit

```

Kết quả chạy



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

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	2	1	6	4	5	3	2097162	0
268501024	0	0	0	0	0	0	0	0
268501056	0	0	0	0	0	0	0	0
268501088	0	0	0	0	0	0	0	0
268501120	0	0	0	0	0	0	0	0
268501152	0	0	0	0	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	0	0	0	0	0	0	0	0
268501248	0	0	0	0	0	0	0	0
268501280	0	0	0	0	0	0	0	0
268501312	0	0	0	0	0	0	0	0
268501344	0	0	0	0	0	0	0	0
268501376	0	0	0	0	0	0	0	0
268501408	0	0	0	0	0	0	0	0

Qua từng vòng lặp

Loop 1	Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)
	268500992	1	2	6	4	5	3
Loop 2	Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)
	268500992	1	2	4	6	5	3
Loop 3	Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)
	268500992	1	2	4	5	6	3
Loop 4	Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)
	268500992	1	2	4	5	3	6
Loop 5	Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)
	268500992	1	2	4	3	5	6
Loop 6	Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)
	268500992	1	2	3	4	5	6

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

→ Đúng với lý thuyết

Chạy chương trình với các bộ test khác nhau

Case: Mảng đầu vào đã được sắp xếp theo thứ tự giảm dần (6, 5, 4, 3, 2, 1)

Messages

Run I/O

Clear

```

5 6 4 3 2 1
5 4 6 3 2 1
5 4 3 6 2 1
5 4 3 2 6 1
5 4 3 2 1 6
4 5 3 2 1 6
4 3 5 2 1 6
4 3 2 5 1 6
4 3 2 1 5 6
3 4 2 1 5 6
3 2 4 1 5 6
3 2 1 4 5 6
2 3 1 4 5 6
2 1 3 4 5 6
1 2 3 4 5 6

-- program is finished running (0) --

```

Case: Mảng đầu vào không có phần tử nào

Messages

Run I/O

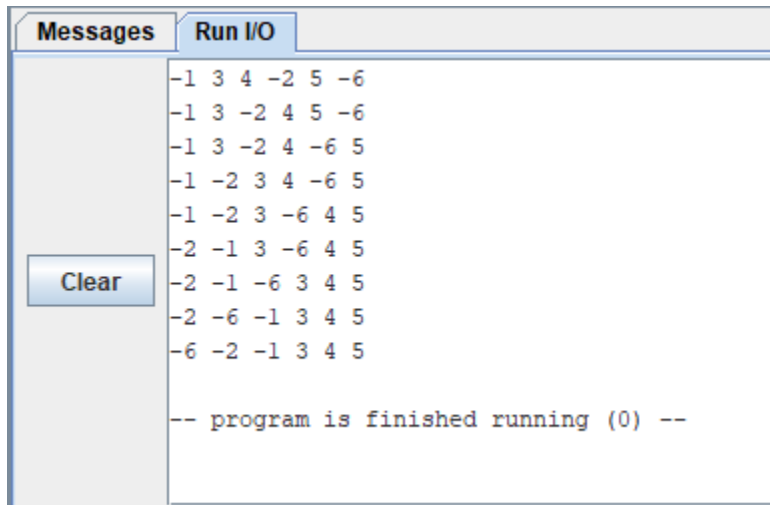
Clear

```

Khong co phan tu nao de sap xep
-- program is finished running (0) --

```

Case: Mảng chứa số dương và âm



```
Messages Run I/O
-1 3 4 -2 5 -6
-1 3 -2 4 5 -6
-1 3 -2 4 -6 5
-1 -2 3 4 -6 5
-1 -2 3 -6 4 5
-2 -1 3 -6 4 5
-2 -1 -6 3 4 5
-2 -6 -1 3 4 5
-6 -2 -1 3 4 5

-- program is finished running (0) --
```

Assignment 4

Viết chương trình thực hiện thuật toán sắp xếp chèn (insertion sort).

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.

Nhập chương trình

```
.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

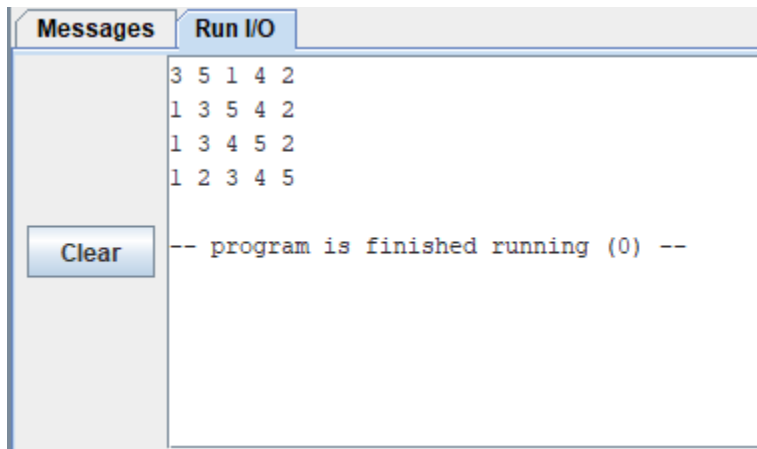
```

Kết quả khi mới khởi tạo mảng:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)
268500992	3	5	1	4	2

Qua từng vòng lặp

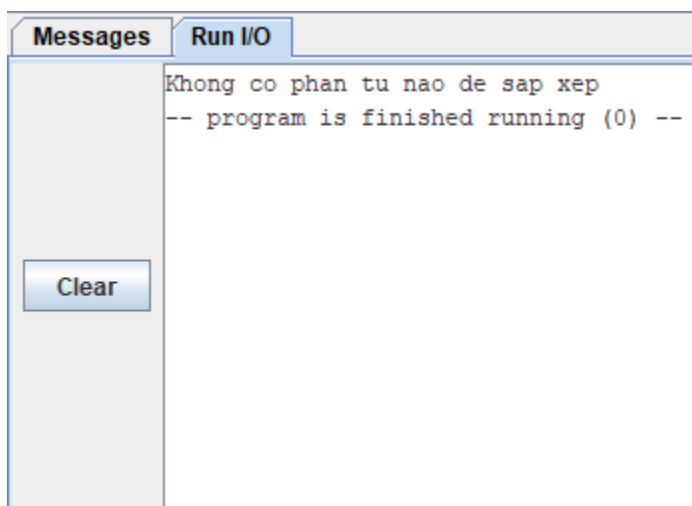
Loop 1	Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)
	268500992	1	3	5	4	2
Loop 2	Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)
	268500992	1	3	4	5	2
Loop 3	Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)
	268500992	1	2	3	4	5



→ Đúng với lý thuyết

Chạy chương trình với các bộ test khác nhau

Case: Mảng không có phần tử nào



Case: Mảng đầu vào đã được sắp xếp theo thứ tự giảm dần (6, 5, 4, 3, 2, 1)

