

BÁO CÁO THỨC HÀNH KIẾN TRÚC MÁY TÍNH TUẦN 6

Họ và tên: Phan Khánh Vũ

MSSV: 20235880

1. Assignment 1:

Nhập chương trình:

```
.data
A: .word -2 1 5 2 -5  # Mảng số nguyên

.text

main:

    la    a0, A        # Lấy địa chỉ của mảng
    li    a1, 20        # Số byte của mảng (5 phần tử, mỗi phần tử 4 byte)
    j     mspfx         # Nhảy đến hàm tính tổng tiền tố

continue:

exit:

    li    a7, 10        # Lờ gọi hệ thống để kết thúc chương trình
    ecall

mspfx:

    li    s0, 0         # Khởi tạo độ dài tổng tiền tố lớn nhất
    li    s1, 0x80000000 # Giá trị nhỏ nhất ban đầu (tương đương -∞)
    li    t0, 0         # Biến đếm (chỉ mục mảng, tính theo byte)
    li    t1, 0         # Tổng tiền tố hiện tại

loop:

    add    t3, a0, t0    # Lấy địa chỉ của phần tử tiếp theo
    lw     t4, 0(t3)     # Đọc giá trị của A[i] từ bộ nhớ
    add    t1, t1, t4    # Cộng giá trị vào tổng tiền tố hiện tại
    blt    s1, t1, mdfy  # Nếu tổng hiện tại lớn hơn tổng max, cập nhật
    j      next         # Nếu không cập nhật, tiếp tục vòng lặp
```

mdfy:

```
srli s0, t0, 2    # Chia 4 để lấy số phần tử
```

```
addi s1, t1, 0    # Cập nhật tổng tiền tố lớn nhất
```

next:

```
addi t0, t0, 4    # Tăng biến đếm lên 4 (vì mỗi phần tử 4 byte)
```

```
blt t0, a1, loop  # Kiểm tra điều kiện dừng vòng lặp
```

done:

```
j continue
```

msofx_end:

- Các bước thực hiện của chương trình:

- + Cùm .data để khởi tạo giá trị cho mảng
- + Cùm câu lệnh đặt trong thẻ main để lấy giá trị của mảng và khởi tạo số byte của mảng, sau đó gọi đến hàm tính tổng tiền tố.
- + Cùm exit sau khi đã thực hiện xong hàm tính tổng tiền tố chương trình quay lại đây và kết thúc chương trình
- + Cùm câu lệnh li để khởi tạo các giá trị sắp được sử dụng
- + Cùm câu lệnh trong thẻ loop để tính tổng và cập nhật giá trị của tổng max khi lớn hơn giá trị của max. Sau đó quay lại thẻ continue và kết thúc chương trình

- Chương trình:

The screenshot displays a MIPS assembly editor window titled 'Lab06_1.asm'. The code is as follows:

```
1 .data
2 A: .word -2 1 5 2 -5    # Mảng số nguyên
3
4 .text
5 main:
6     la a0, A             # Lấy địa chỉ của mảng
7     li a1, 20            # Số byte của mảng (5 phần tử, mỗi phần tử 4 byte)
8     jalr a0, a1          # Nhảy đến hàm tính tổng tiền tố
9
10    continue
11 exit:
12     li a0, 10           # Mã gọi hệ thống để kết thúc chương trình
13     syscall
14
15    msofx:
16     li a0, 0            # Khởi tạo biến đếm tổng tiền tố lớn nhất
17     li a1, 0x80000000    # Giá trị nhỏ nhất ban đầu (trong khoảng -2^31)
18     li t0, 0            # Biến đếm (chỉ mục mảng, tính theo byte)
19     li t1, 0            # Tổng tiền tố hiện tại
20
21    loop:
22     addi t3, a0, t0      # Lấy địa chỉ của phần tử tiếp theo
23     lw t4, 0(t3)         # Đọc giá trị của A[i] từ bộ nhớ
24     addi t1, t1, t4      # Cộng giá trị vào tổng tiền tố hiện tại
25     blt t4, t1, msofx    # Nếu tổng hiện tại lớn hơn tổng max, cập nhật
26
27     j loop              # Nếu không cập nhật, tiếp tục vòng lặp
```

The right panel shows the register table with the following values:

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
rp	2	0x00000000
rp	3	0x00000000
t0	4	0x00000001
t1	5	0x00000000
t2	6	0x00000000
t3	7	0x00000000
t4	8	0x00000000
t5	9	0x00000000
t6	10	0x00000000
t7	11	0x00000001
t8	12	0x00000000
t9	13	0x00000000
t10	14	0x00000000
t11	15	0x00000000
t12	16	0x00000000
t13	17	0x00000000
t14	18	0x00000000
t15	19	0x00000000
t16	20	0x00000000
t17	21	0x00000000
t18	22	0x00000000
t19	23	0x00000000
t20	24	0x00000000
t21	25	0x00000000
t22	26	0x00000000
t23	27	0x00000000
t24	28	0x00000001
t25	29	0x00000000
t26	30	0x00000000
t27	31	0x00000000
pc		0x00400001

- Quan sát sự thay đổi của thanh ghi:

Code	Basic	Source	
0x01c10517	auipc x10,0x0000fc10	6: la a0, A	# Lấy địa chỉ của mảng
0x00050513	addi x10,x10,0		
0x01400593	addi x11,x0,20	7: li a1, 20	# Số byte của mảng (5..
0x00c0006f	jal x0,0x0000000c	8: j mspfx	# Nhảy đến hàm tính t...
0x00a00893	addi x17,x0,10	12: li a7, 10	# Lỗi gọi hệ thống để...
0x00000073	ecall	13: ecall	
0x00000413	addi x8,x0,0	16: li s0, 0	# Khởi tạo độ dài tồn...
0x800004b7	lui x9,0xffff80000	17: li s1, 0x80000000	# Giá trị nhỏ nhất ba...
0x00048493	addi x9,x9,0		
0x00000293	addi x5,x0,0	18: li t0, 0	# Biến đếm (chỉ mục m...
0x01c10517	auipc x10,0x0000fc10	6: la a0, A	# Lấy địa chỉ của mảng
0x00050513	addi x10,x10,0		
0x01400593	addi x11,x0,20	7: li a1, 20	# Số byte của mảng (5..
0x00c0006f	jal x0,0x0000000c	8: j mspfx	# Nhảy đến hàm tính t...
0x00a00893	addi x17,x0,10	12: li a7, 10	# Lỗi gọi hệ thống để...
0x00000073	ecall	13: ecall	
0x00000413	addi x8,x0,0	16: li s0, 0	# Khởi tạo độ dài tồn...
0x800004b7	lui x9,0xffff80000	17: li s1, 0x80000000	# Giá trị nhỏ nhất ba...
0x00048493	addi x9,x9,0		
0x00000293	addi x5,x0,0	18: li t0, 0	# Biến đếm (chỉ mục m...
Code	Basic	Source	
0x01c10517	auipc x10,0x0000fc10	6: la a0, A	# Lấy địa chỉ của mảng
0x00050513	addi x10,x10,0		
0x01400593	addi x11,x0,20	7: li a1, 20	# Số byte của mảng (5..
0x00c0006f	jal x0,0x0000000c	8: j mspfx	# Nhảy đến hàm tính t...
0x00a00893	addi x17,x0,10	12: li a7, 10	# Lỗi gọi hệ thống để...
0x00000073	ecall	13: ecall	
0x00000413	addi x8,x0,0	16: li s0, 0	# Khởi tạo độ dài tồn...
0x800004b7	lui x9,0xffff80000	17: li s1, 0x80000000	# Giá trị nhỏ nhất ba...
0x00048493	addi x9,x9,0		
0x00000293	addi x5,x0,0	18: li t0, 0	# Biến đếm (chỉ mục m...
Code	Basic	Source	
0x800004b7	lui x9,0xffff80000	17: li s1, 0x80000000	# Giá trị nhỏ nhất ba...
0x00048493	addi x9,x9,0		
0x00000293	addi x5,x0,0	18: li t0, 0	# Biến đếm (chỉ mục m...
0x00000313	addi x6,x0,0	19: li t1, 0	# Tổng tiền tổ hiện tại
0x00550e33	add x28,x10,x5	22: add t3, a0, t0	# Lấy địa chỉ của phả...
0x000e2e83	lw x29,0(x28)	23: lw t4, 0(t3)	# Đọc giá trị của A[i...
0x01d30333	add x6,x6,x29	24: add t1, t1, t4	# Cộng giá trị vào tổ...
0x0064c463	blt x9,x6,0x000000008	25: blt s1, t1, mdfy	# Nếu tổng hiện tại l...
0x00c0006f	jal x0,0x0000000c	27: j next	# Nếu không cập nhật...
0x0022d413	srlrli x8,s0,2	30: srlrli s0, t0, 2	# Chia 4 để lấy số nh...
Code	Basic	Source	
0x800004b7	lui x9,0xffff80000	17: li s1, 0x80000000	# Giá trị nhỏ nhất ba...
0x00048493	addi x9,x9,0		
0x00000293	addi x5,x0,0	18: li t0, 0	# Biến đếm (chỉ mục m...
0x00000313	addi x6,x0,0	19: li t1, 0	# Tổng tiền tổ hiện tại
0x00550e33	add x28,x10,x5	22: add t3, a0, t0	# Lấy địa chỉ của phả...
0x000e2e83	lw x29,0(x28)	23: lw t4, 0(t3)	# Đọc giá trị của A[i...
0x01d30333	add x6,x6,x29	24: add t1, t1, t4	# Cộng giá trị vào tổ...
0x0064c463	blt x9,x6,0x000000008	25: blt s1, t1, mdfy	# Nếu tổng hiện tại l...
0x00c0006f	jal x0,0x0000000c	27: j next	# Nếu không cập nhật...
0x0022d413	srlrli x8,s0,2	30: srlrli s0, t0, 2	# Chia 4 để lấy số ph...
Code	Basic	Source	
0x800004b7	lui x9,0xffff80000	17: li s1, 0x80000000	# Giá trị nhỏ nhất ba...
0x00048493	addi x9,x9,0		
0x00000293	addi x5,x0,0	18: li t0, 0	# Biến đếm (chỉ mục m...
0x00000313	addi x6,x0,0	19: li t1, 0	# Tổng tiền tổ hiện tại
0x00550e33	add x28,x10,x5	22: add t3, a0, t0	# Lấy địa chỉ của phả...
0x000e2e83	lw x29,0(x28)	23: lw t4, 0(t3)	# Đọc giá trị của A[i...
0x01d30333	add x6,x6,x29	24: add t1, t1, t4	# Cộng giá trị vào tổ...
0x0064c463	blt x9,x6,0x000000008	25: blt s1, t1, mdfy	# Nếu tổng hiện tại l...
0x00c0006f	jal x0,0x0000000c	27: j next	# Nếu không cập nhật...
0x0022d413	srlrli x8,s0,2	30: srlrli s0, t0, 2	# Chia 4 để lấy số ph...
Code	Basic	Source	
0x00048493	addi x9,x9,0		
0x00000293	addi x5,x0,0	18: li t0, 0	# Biến đếm (chỉ mục m...
0x00000313	addi x6,x0,0	19: li t1, 0	# Tổng tiền tổ hiện tại
0x00550e33	add x28,x10,x5	22: add t3, a0, t0	# Lấy địa chỉ của phả...
0x000e2e83	lw x29,0(x28)	23: lw t4, 0(t3)	# Đọc giá trị của A[i...
0x01d30333	add x6,x6,x29	24: add t1, t1, t4	# Cộng giá trị vào tổ...
0x0064c463	blt x9,x6,0x000000008	25: blt s1, t1, mdfy	# Nếu tổng hiện tại l...
0x00c0006f	jal x0,0x0000000c	27: j next	# Nếu không cập nhật...
0x0022d413	srlrli x8,s0,2	30: srlrli s0, t0, 2	# Chia 4 để lấy số ph...
0x00030493	addi x9,x6,0	31: addi s1, t1, 0	# Cập nhật tổng tiền ...
Code	Basic	Source	
0x00000293	addi x5,x0,0	18: li t0, 0	# Biến đếm (chỉ mục m...
0x00000313	addi x6,x0,0	19: li t1, 0	# Tổng tiền tổ hiện tại
0x00550e33	add x28,x10,x5	22: add t3, a0, t0	# Lấy địa chỉ của phả...
0x000e2e83	lw x29,0(x28)	23: lw t4, 0(t3)	# Đọc giá trị của A[i...
0x01d30333	add x6,x6,x29	24: add t1, t1, t4	# Cộng giá trị vào tổ...
0x0064c463	blt x9,x6,0x000000008	25: blt s1, t1, mdfy	# Nếu tổng hiện tại l...
0x00c0006f	jal x0,0x0000000c	27: j next	# Nếu không cập nhật...
0x0022d413	srlrli x8,s0,2	30: srlrli s0, t0, 2	# Chia 4 để lấy số ph...
0x00030493	addi x9,x6,0	31: addi s1, t1, 0	# Cập nhật tổng tiền ...
0x00428293	addi x5,x5,4	34: addi t0, t0, 4	# Tăng biến đếm lên 4...
sp		2	0x7ffffcfe
gp		3	0x10008000
tp		4	0x00000000
t0		5	0x00000000
t1		6	0x00000000
t2		7	0x00000000
s0		8	0x00000000
s1		9	0x00000000
a0		10	0x10010000
a1		11	0x00000000
a2		12	0x00000000
sp		2	0x7ffffcfe
gp		3	0x10008000
tp		4	0x00000000
t0		5	0x00000000
t1		6	0x00000000
t2		7	0x00000000
s0		8	0x00000000
s1		9	0x00000000
a0		10	0x10010000
a1		11	0x00000014
a2		12	0x00000000
sp		2	0x7ffffcfe
gp		3	0x10008000
tp		4	0x00000000
t0		5	0x00000000
t1		6	0x00000000
t2		7	0x00000000
s0		8	0x00000000
s1		9	0x80000000
a0		10	0x10010000
a1		11	0x00000014
a2		12	0x00000000
sp		2	0x7ffffcfe
gp		3	0x10008000
tp		4	0x00000000
t0		5	0x00000000
t1		6	0xffffffff
t2		7	0x00000000
s0		8	0x00000000
s1		9	0x80000000
a0		10	0x10010000
a1		11	0x00000014
a2		12	0x00000000
sp		2	0x7ffffcfe
gp		3	0x10008000
tp		4	0x00000000
t0		5	0x00000000
t1		6	0xffffffff
t2		7	0x00000000
s0		8	0x00000000
s1		9	0x80000000
a0		10	0x10010000
a1		11	0x00000014
a2		12	0x00000000
sp		2	0x7ffffcfe
gp		3	0x10008000
tp		4	0x00000000
t0		5	0x00000000
t1		6	0xffffffff
t2		7	0x00000000
s0		8	0x00000000
s1		9	0x80000000
a0		10	0x10010000
a1		11	0x00000014
a2		12	0x00000000
sp		2	0x7ffffcfe
gp		3	0x10008000
tp		4	0x00000000
t0		5	0x00000000
t1		6	0xffffffff
t2		7	0x00000000
s0		8	0x00000000
s1		9	0x80000000
a0		10	0x10010000
a1		11	0x00000014
a2		12	0x00000000

- Sau đó vòng lặp sẽ tiếp tục, theo lý thuyết Chương trình sẽ tìm ra tổng tiền tố lớn nhất là 6, được lưu trong s1. Output mong đợi của chương trình:
max_prefix_sum = 6

Kiểm tra kết quả thực hiện chương trình:

+ Với giá trị của mảng được khởi tạo là -2 1 5 2 -5 thì kết quả là:

Registers	Floating Point	Control and Status	
Name	Number	Value	
zero	0	0x00000000	
ra	1	0x00000000	
sp	2	0x7ffefffc	
gp	3	0x10008000	
tp	4	0x00000000	
t0	5	0x00000014	
t1	6	0x00000001	
t2	7	0x00000000	
s0	8	0x00000003	
s1	9	0x00000006	
a0	10	0x10010000	
a1	11	0x00000014	

+ Giá trị được lưu ở của sổ data segment:

Data Segment					
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x10010000	0xfffffffffe	0x00000001	0x00000005	0x00000002	0xfffffffffb
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

⇒ Chương trình chạy đúng với thuật toán.

2. Assignment 2:

Nhập chương trình:

.data

A: .word 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

Aend: .word

space: .asciz " "

newline: .asciz "\n" # Xuống dòng sau mỗi lần in mảng

.text

main:

```

la    a0, A      # Lưu địa chỉ của mảng A
la    a1, Aend
addi  a1, a1, -4 # Địa chỉ của phần tử cuối cùng trong mảng
li    s2 40      # Khởi tạo số byte của mảng

j     sort      # sort
after_sort:
li    a7, 10
ecall
end_main:

sort:
beq   a0, a1, done # Kiểm tra nếu mảng có 1 kí tự thì kết thúc
j     max        # Gọi đến hàm tìm max

after_max:
lw    t0, 0(a1)    # Lấy giá trị của phần tử cuối cùng
sw    t0, 0(s0)    # Thay giá trị của phần tử cuối cùng bằng phần tử
max
sw    s1, 0(a1)    # Đổi giá trị của phần tử lớn nhất là phần tử cuối
vừa trở
addi  a1, a1, -4   # Lùi còn trở đến phần tử trước đó
addi  s5, a0, 0
li    s3 0        # Khởi tạo biến đếm
arrLoop:
add   s4, s3, s5
lw    a0, 0(s4)
li    a7, 1
ecall

# In khoảng trắng
la    a0, space
li    a7, 4
ecall

addi  s3, s3, 4    # Tăng biến đếm
bge   s3, s2, endArrLoop # Kiểm tra điều kiện dừng
j     arrLoop      # Tiếp tục vòng lặp
endArrLoop:
la    a0, newline

```

```

li a7, 4
ecall

addi a0, s5, 0
j sort      # Gọi đến hàm sort và tiếp tục thực hiện
done:
j after_sort
max:
addi s0, a0, 0 # Lấy địa chỉ của phần tử đầu tiên trong mảng
lw s1, 0(s0) # Lấy giá trị của phần tử có vị trí s0
addi t0, a0, 0 # Con trỏ trỏ đến phần tử đầu tiên
loop:
beq t0, a1, ret # kiểm tra để kết thúc chương trình
addi t0, t0, 4 # Tăng biến đếm
lw t1, 0(t0) # load next element into t1
blt t1, s1, loop # kiểm tra để tiếp tục vòng lặp
addi s0, t0, 0 # Địa chỉ mới của max
addi s1, t1, 0 # Giá trị mới của max
j loop
# Sau khi đổi tiếp tục thực hiện chương trình
ret:
j after_max

```

- **Các bước thực hiện của chương trình:**

- + Cụm câu lệnh .data để khai báo giá trị của mảng và các chuỗi kí tự
- + Cụm main để khởi tạo các giá trị của chương trình và gọi đến hàm sort
- + Cụm lệnh sau thẻ after_sort để gọi đến hệ thống kết thúc chương trình
- + Cụm lệnh sau thẻ after_max là vòng lặp ngoài để thay đổi vị trí con trỏ, trỏ đến vị trí tiếp theo được chọn, rồi gọi đến j max
- + Cụm lệnh trong thẻ arrLoop để in giá trị của các phần tử trong mảng sau mỗi vòng lặp, sau đó thực hiện xuống dòng và tiếp tục nhảy đến thẻ after_sort
- + Cụm câu lệnh trong thẻ max để lưu vị trí con trỏ trỏ đến phần tử đầu tiên trong mảng
- + Câu lệnh trong cụm loop để tìm xem giá trị lớn nhất trong phần còn lại của mảng đang xét, sau khi duyệt hết thì quay lại after_max và tiếp tục thực hiện chương trình.

- **Chạy chương trình:**

Registers	Floating Point	Control and Status
Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000002
t1	6	0x00000002
t2	7	0x00000000
s0	8	0x10010004
s1	9	0x00000002
a0	10	0x10010000
a1	11	0x10010000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x0000000a
s2	18	0x00000028
s3	19	0x00000028
s4	20	0x10010024
s5	21	0x10010000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400024

Các dữ liệu được lưu ở cửa sổ data segment là:

Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007	0x00000008
0x00000009	0x0000000a	0x0000000b	0x0000000c	0x0000000d	0x0000000e	0x0000000f	0x00000010
0x00000011	0x00000012	0x00000013	0x00000014	0x00000015	0x00000016	0x00000017	0x00000018
0x00000019	0x0000001a	0x0000001b	0x0000001c	0x0000001d	0x0000001e	0x0000001f	0x00000020
0x00000021	0x00000022	0x00000023	0x00000024	0x00000025	0x00000026	0x00000027	0x00000028
0x00000029	0x0000002a	0x0000002b	0x0000002c	0x0000002d	0x0000002e	0x0000002f	0x00000030
0x00000031	0x00000032	0x00000033	0x00000034	0x00000035	0x00000036	0x00000037	0x00000038
0x00000039	0x0000003a	0x0000003b	0x0000003c	0x0000003d	0x0000003e	0x0000003f	0x00000040
0x00000041	0x00000042	0x00000043	0x00000044	0x00000045	0x00000046	0x00000047	0x00000048
0x00000049	0x0000004a	0x0000004b	0x0000004c	0x0000004d	0x0000004e	0x0000004f	0x00000050
0x00000051	0x00000052	0x00000053	0x00000054	0x00000055	0x00000056	0x00000057	0x00000058
0x00000059	0x0000005a	0x0000005b	0x0000005c	0x0000005d	0x0000005e	0x0000005f	0x00000060
0x00000061	0x00000062	0x00000063	0x00000064	0x00000065	0x00000066	0x00000067	0x00000068
0x00000069	0x0000006a	0x0000006b	0x0000006c	0x0000006d	0x0000006e	0x0000006f	0x00000070
0x00000071	0x00000072	0x00000073	0x00000074	0x00000075	0x00000076	0x00000077	0x00000078
0x00000079	0x0000007a	0x0000007b	0x0000007c	0x0000007d	0x0000007e	0x0000007f	0x00000080
0x00000081	0x00000082	0x00000083	0x00000084	0x00000085	0x00000086	0x00000087	0x00000088
0x00000089	0x0000008a	0x0000008b	0x0000008c	0x0000008d	0x0000008e	0x0000008f	0x00000090
0x00000091	0x00000092	0x00000093	0x00000094	0x00000095	0x00000096	0x00000097	0x00000098
0x00000099	0x0000009a	0x0000009b	0x0000009c	0x0000009d	0x0000009e	0x0000009f	0x000000a0
0x000000a1	0x000000a2	0x000000a3	0x000000a4	0x000000a5	0x000000a6	0x000000a7	0x000000a8
0x000000a9	0x000000aa	0x000000ab	0x000000ac	0x000000ad	0x000000ae	0x000000af	0x000000b0
0x000000b1	0x000000b2	0x000000b3	0x000000b4	0x000000b5	0x000000b6	0x000000b7	0x000000b8
0x000000b9	0x000000ba	0x000000bb	0x000000bc	0x000000bd	0x000000be	0x000000bf	0x000000c0
0x000000c1	0x000000c2	0x000000c3	0x000000c4	0x000000c5	0x000000c6	0x000000c7	0x000000c8
0x000000c9	0x000000ca	0x000000cb	0x000000cc	0x000000cd	0x000000ce	0x000000cf	0x000000d0
0x000000d1	0x000000d2	0x000000d3	0x000000d4	0x000000d5	0x000000d6	0x000000d7	0x000000d8
0x000000d9	0x000000da	0x000000db	0x000000dc	0x000000dd	0x000000de	0x000000df	0x000000e0
0x000000e1	0x000000e2	0x000000e3	0x000000e4	0x000000e5	0x000000e6	0x000000e7	0x000000e8
0x000000e9	0x000000ea	0x000000eb	0x000000ec	0x000000ed	0x000000ee	0x000000ef	0x000000f0
0x000000f1	0x000000f2	0x000000f3	0x000000f4	0x000000f5	0x000000f6	0x000000f7	0x000000f8
0x000000f9	0x000000fa	0x000000fb	0x000000fc	0x000000fd	0x000000fe	0x000000ff	0x00000100
0x00000101							

3. Assignment 3 (bubble sort).

- **Nhập chương trình: (sắp xếp tăng dần)**

```
.data
A: .word 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

.text
main:

    la    a0, A        # Lấy địa chỉ của mảng A
    li    s2 40         # Lấy số byte của mảng A
    li    s0 0          # Khởi tạo biến đếm Loop1
```

```
A: .word 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

.text

main:

    la    a0, A        # Lấy địa chỉ của mảng A
    li    s2, 40        # Lấy số byte của mảng A
    li    s0, 0         # Khởi tạo biến đếm Loop1
```

```
.text
main:
    la    a0, A        # Lấy địa chỉ của mảng A
    li    s2, 40        # Lấy số byte của mảng A
    li    s0, 0         # Khởi tạo biến đếm Loop1
```

```
main:
    la    a0, A        # Lấy địa chỉ của mảng A
    li    s2, 40        # Lấy số byte của mảng A
    li    s0, 0         # Khởi tạo biến đếm Loop1
```

la	a0, A	# Lấy địa chỉ của mảng A
li	s2 40	# Lấy số byte của mảng A
li	s0 0	# Khởi tạo biến đếm Loop1

li	s2 40	# Lấy số byte của mảng A
li	s0 0	# Khởi tạo biến đếm Loop1

```
li      s0 0      # Khởi tạo biến đếm Loop1
```

Loop1:

bge s0, s2, endLoop1# Kiểm tra điều kiện dừng của Loop1

addi s1, s0, 4 # Biến đếm loop 2

Loop2:

bge s1, s2, endLoop2

Kiểm tra điều kiện dừng Loop2

add s4, a0, s0 # Lấy địa chỉ của A[i]

lw a2, 0(s4) # Lấy giá trị của A[i]

add s5, a0, s1 # Lấy địa chỉ của A[j]

lw a3, 0(s5) # Lấy giá trị của A[j]

blt a3, a2, swap # So sánh để tìm phần tử lớn

j continue

swap:

Đổi chỗ A[i] và A[j]

sw a3, 0(s4)

sw a2, 0(s5)

continue:

addi s1, s1, 4 # Tăng biến đếm Loop2

j Loop2

endLoop2:

addi s0, s0, 4 # Tăng biến đếm Loop1

j Loop1

endLoop1:

- **Các bước thực hiện của chương trình:**

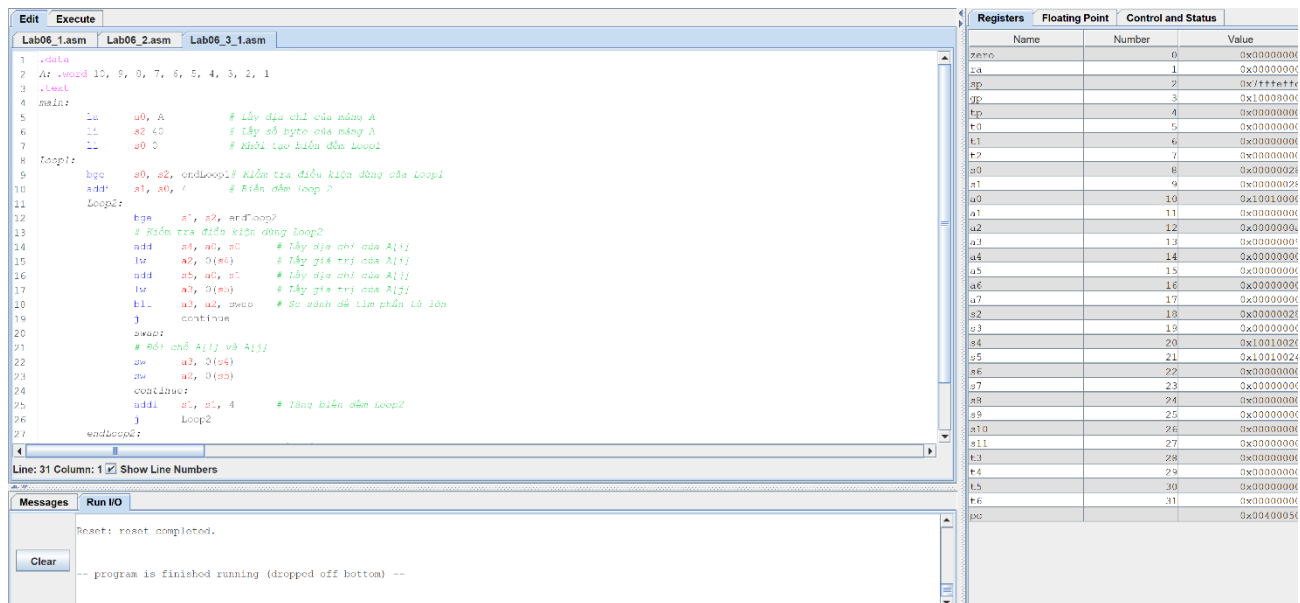
+ Cụm câu lệnh .data để khởi tạo giá trị của các phần tử trong mảng

+ Cụm câu lệnh trong thẻ main để khởi tạo độ dài mảng A và biến đếm của Loop1

+ Cụm câu lệnh trong thẻ Loop1 để thực hiện loop2, khởi tạo biến đếm cho loop2

- + Cụm câu lệnh trong Loop2 để kiểm tra và so sánh A[i] và A[j] xem nếu A[i] > A[j] thì swap hay giá trị, nếu không thì tiếp tục lặp
- + Chương trình kết thúc khi đã thực hiện lặp đến hết các phần tử trong mảng với độ phức tạp $O(n^2)$

- Chạy chương trình:



- Kết quả thực hiện chương trình:

- + Khởi tạo giá trị cho mảng lần lượt là 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
- + Giá trị ban đầu của cửa sổ Data segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000000a	0x00000009	0x00000008	0x00000007	0x00000006	0x00000005	0x00000004	0x00000003
0x10010020	0x00000002	0x00000001	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

- + Giá trị của cửa sổ Data segment sau khi thực hiện chương trình là:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007	0x00000008
0x10010020	0x00000009	0x0000000a	0x0000000b	0x0000000c	0x0000000d	0x0000000e	0x0000000f	0x00000010
0x10010040	0x00000011	0x00000012	0x00000013	0x00000014	0x00000015	0x00000016	0x00000017	0x00000018
0x10010060	0x00000019	0x0000001a	0x0000001b	0x0000001c	0x0000001d	0x0000001e	0x0000001f	0x00000020
0x10010080	0x00000021	0x00000022	0x00000023	0x00000024	0x00000025	0x00000026	0x00000027	0x00000028
0x100100a0	0x00000029	0x0000002a	0x0000002b	0x0000002c	0x0000002d	0x0000002e	0x0000002f	0x00000030
0x100100c0	0x00000031	0x00000032	0x00000033	0x00000034	0x00000035	0x00000036	0x00000037	0x00000038
0x100100e0	0x00000039	0x0000003a	0x0000003b	0x0000003c	0x0000003d	0x0000003e	0x0000003f	0x00000040
0x10010100	0x00000041	0x00000042	0x00000043	0x00000044	0x00000045	0x00000046	0x00000047	0x00000048
0x10010120	0x00000049	0x0000004a	0x0000004b	0x0000004c	0x0000004d	0x0000004e	0x0000004f	0x00000050
0x10010140	0x00000051	0x00000052	0x00000053	0x00000054	0x00000055	0x00000056	0x00000057	0x00000058
0x10010160	0x00000059	0x0000005a	0x0000005b	0x0000005c	0x0000005d	0x0000005e	0x0000005f	0x00000060
0x10010180	0x00000061	0x00000062	0x00000063	0x00000064	0x00000065	0x00000066	0x00000067	0x00000068
0x100101a0	0x00000069	0x0000006a	0x0000006b	0x0000006c	0x0000006d	0x0000006e	0x0000006f	0x00000070

- **Nhập chương trình: (sắp xếp giảm dần)**

```
.data
    A: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

.text
main:
    la    a0, A        # Lấy địa chỉ của mảng A
    li    s2 40         # Lấy số byte của mảng A
    li    s0 0          # Khởi tạo biến đếm Loop1

Loop1:
    bge   s0, s2, endLoop1 # Kiểm tra điều kiện dừng của Loop1
    addi  s1, s0, 4       # Biến đếm loop 2
    Loop2:
        bge   s1, s2, endLoop2
        # Kiểm tra điều kiện dừng Loop2
        add   s4, a0, s0   # Lấy địa chỉ của A[i]
        lw    a2, 0(s4)    # Lấy giá trị của A[i]
        add   s5, a0, s1   # Lấy địa chỉ của A[j]
        lw    a3, 0(s5)    # Lấy giá trị của A[j]
        blt   a2, a3, swap  # So sánh để tìm phần tử lớn
        j     continue
    swap:
        # Đổi chỗ A[i] và A[j]
        sw    a3, 0(s4)
        sw    a2, 0(s5)
    continue:
        addi  s1, s1, 4     # Tăng biến đếm Loop2
        j     Loop2
```

```

endLoop2:

    addi    s0, s0, 4    # Tăng biến đếm Loop1

    j       Loop1

endLoop1:

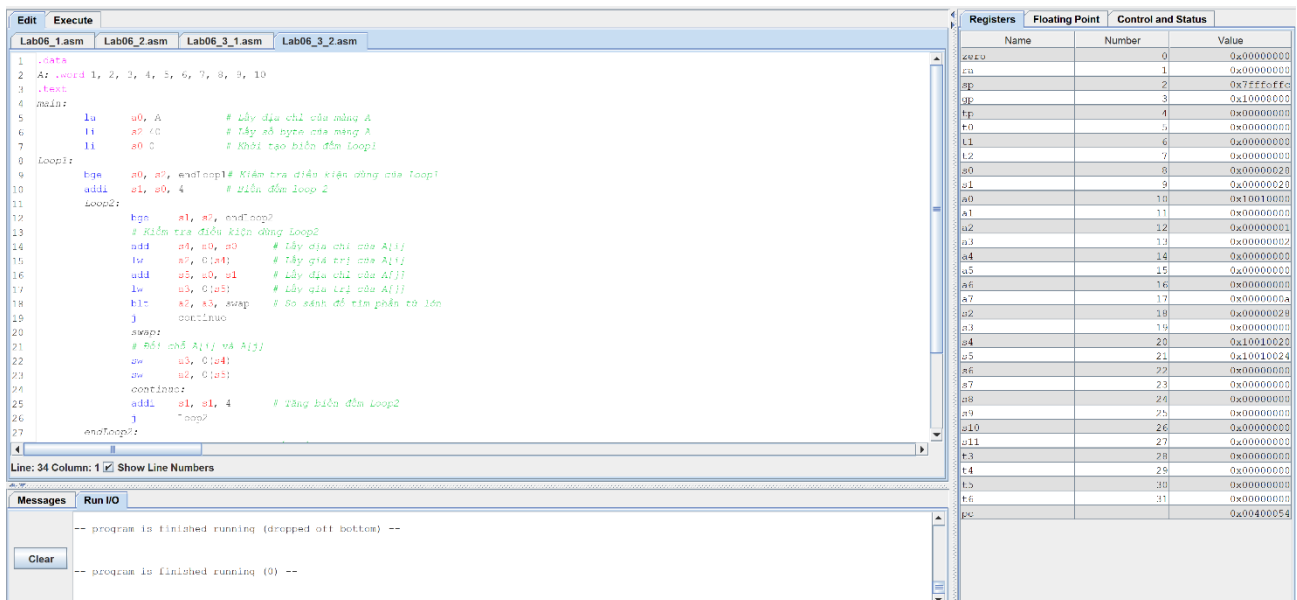
    li      a7 10

    ecall

endMain:

```

- **Các bước thực hiện chương trình:**
 - + Câu lệnh điều kiện để swap A[i] và A[j] thay đổi so với chương trình sắp xếp tăng dần.
- **Chạy chương trình:**



- **Kết quả của chương trình:**
 - + Với giá trị của mảng A là 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
 - + Giá trị ban đầu của cửa sổ Data segment là:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007	0x00000008
0x10010020	0x00000009	0x0000000a	0x0000000b	0x0000000c	0x0000000d	0x0000000e	0x0000000f	0x00000010
0x10010040	0x00000011	0x00000012	0x00000013	0x00000014	0x00000015	0x00000016	0x00000017	0x00000018
0x10010060	0x00000019	0x0000001a	0x0000001b	0x0000001c	0x0000001d	0x0000001e	0x0000001f	0x00000020
0x10010080	0x00000021	0x00000022	0x00000023	0x00000024	0x00000025	0x00000026	0x00000027	0x00000028
0x100100a0	0x00000029	0x0000002a	0x0000002b	0x0000002c	0x0000002d	0x0000002e	0x0000002f	0x00000030
0x100100c0	0x00000031	0x00000032	0x00000033	0x00000034	0x00000035	0x00000036	0x00000037	0x00000038
0x100100e0	0x00000039	0x0000003a	0x0000003b	0x0000003c	0x0000003d	0x0000003e	0x0000003f	0x00000040
0x10010100	0x00000041	0x00000042	0x00000043	0x00000044	0x00000045	0x00000046	0x00000047	0x00000048
0x10010120	0x00000049	0x0000004a	0x0000004b	0x0000004c	0x0000004d	0x0000004e	0x0000004f	0x00000050
0x10010140	0x00000051	0x00000052	0x00000053	0x00000054	0x00000055	0x00000056	0x00000057	0x00000058
0x10010160	0x00000059	0x0000005a	0x0000005b	0x0000005c	0x0000005d	0x0000005e	0x0000005f	0x00000060
0x10010180	0x00000061	0x00000062	0x00000063	0x00000064	0x00000065	0x00000066	0x00000067	0x00000068
0x100101a0	0x00000069	0x0000006a	0x0000006b	0x0000006c	0x0000006d	0x0000006e	0x0000006f	0x00000070

+ Giá trị của của số Data segment sau khi thực hiện chương trình là:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000000a	0x00000009	0x00000008	0x00000007	0x00000006	0x00000005	0x00000004	0x00000003
0x10010020	0x00000002	0x00000001	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

⇒ Kết quả đúng với kết quả tính toán.

4. Assignment 4 (insertion sort)

- Nhập chương trình: (sắp xếp tăng dần)

```
.data
```

```
A: .word 1, 3, 5, 7, 9, 10, 2, 4, 6, 8
```

```
.text
```

```
main:
```

```
la a0, A # Lấy địa chỉ của mảng A
```

```
li s2, 40 # Lấy số byte của mảng A
```

```
li s0, 4 # Bắt đầu từ phần tử thứ hai (index 1)
```

```
Loop1:
```

```
bge s0, s2, endLoop1 # Kiểm tra điều kiện dừng của Loop1
```

```
add s1, a0, s0 # Lấy địa chỉ của A[i]
```

```
lw a1, 0(s1) # Lấy giá trị của A[i] (key)
```

```
addi s3, s0, -4 # j = i - 1
```

```
Loop2:
```

```
blt s3, zero, endLoop2 # Nếu j < 0 thì dừng
```

```
add s4, a0, s3 # Lấy địa chỉ của A[j]
```

```
lw a2, 0(s4) # Lấy giá trị của A[j]
```

```
bge a1, a2, shift_right # Nếu A[j] > key thì dời A[j]
j   endLoop2
```

shift_right:

```
addi s5, s3, 4 # Lấy địa chỉ của A[j+1]
add  s6, a0, s5 # Tính địa chỉ trong mảng
sw  a2, 0(s6)  # A[j+1] = A[j]
addi s3, s3, -4 # j--
j   Loop2
```

endLoop2:

```
addi s5, s3, 4 # Lấy địa chỉ của A[j+1]
add  s6, a0, s5 # Tính địa chỉ trong mảng
sw  a1, 0(s6)  # A[j+1] = key
addi s0, s0, 4 # i++
j   Loop1
```

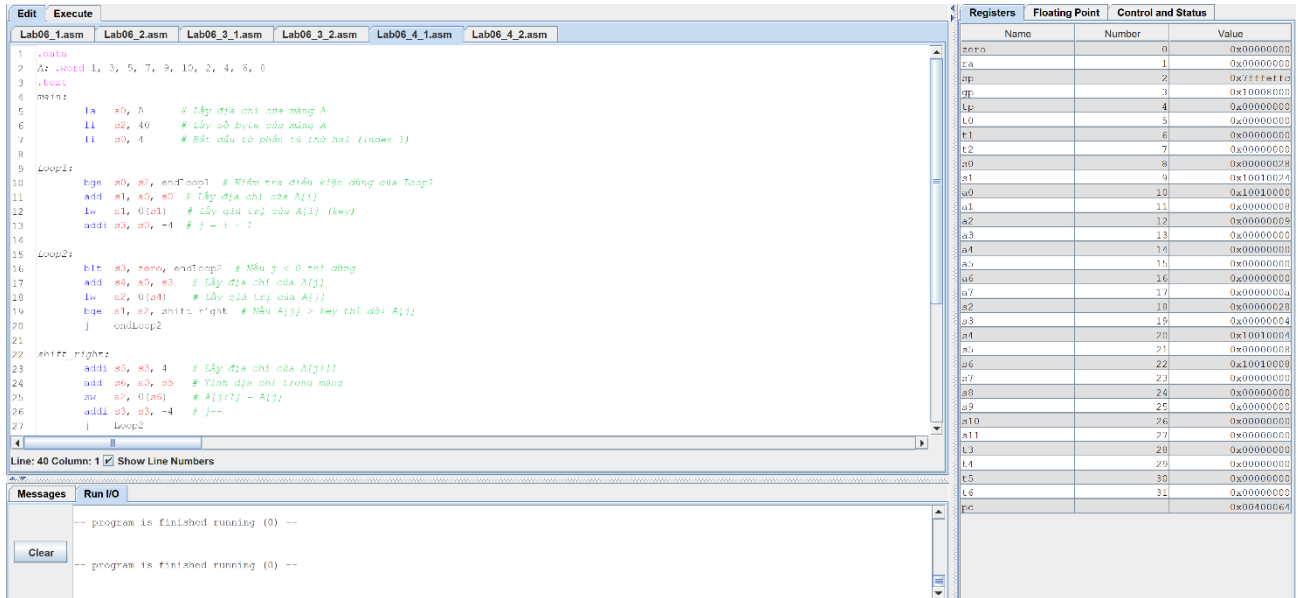
endLoop1:

```
# Kết thúc chương trình
li  a7, 10 # syscall 10 (exit)
ecall
```

- **Các bước thực hiện của chương trình:**

- + Cúm .data để khai báo giá trị của các phần tử trong mảng
- + Cúm main để khởi tạo các giá trị cho vòng lặp Loop1, biến đếm và số byte của mảng.
- + Sau thẻ Loop1 khởi tạo giá trị của biến đếm và bắt đầu Loop2
- + Trong thẻ Loop2 so sánh giá trị của của A[j] và key nếu A[j] > key thì dịch phải nếu không thì kết thúc vòng lặp.
- + Sau khi duyệt đến hết vòng lặp 1 kết thúc chương trình.

- Chạy chương trình:



- Kết quả của chương trình

+ Với giá trị của mảng A là : 1, 3, 5, 7, 9, 10, 2, 4, 6, 8

+ Giá trị ban đầu của cửa sổ Data segment là:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000001	0x00000003	0x00000005	0x00000007	0x00000009	0x0000000a	0x00000002	0x00000004
0x10010020	0x0000000e	0x00000008	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

+ Giá trị của cửa sổ Data segment sau khi thực hiện chương trình là:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000000a	0x00000009	0x00000008	0x00000007	0x00000006	0x00000005	0x00000004	0x00000003
0x10010020	0x00000002	0x00000001	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

- Nhập chương trình: (Sắp xếp giảm dần)

.data

A: .word 1, 3, 5, 7, 9, 10, 2, 4, 6, 8

.text

main:

```
la  a0, A    # Lấy địa chỉ của mảng A
li  s2, 40    # Lấy số byte của mảng A
li  s0, 4     # Bắt đầu từ phần tử thứ hai (index 1)
```

Loop1:

```
bge s0, s2, endLoop1 # Kiểm tra điều kiện dừng của Loop1
add s1, a0, s0 # Lấy địa chỉ của A[i]
lw  a1, 0(s1) # Lấy giá trị của A[i] (key)
addi s3, s0, -4 # j = i - 1
```

Loop2:

```
blt s3, zero, endLoop2 # Nếu j < 0 thì dừng
add s4, a0, s3 # Lấy địa chỉ của A[j]
lw  a2, 0(s4) # Lấy giá trị của A[j]
bge a1, a2, shift_right # Nếu A[j] > key thì dời A[j]
j   endLoop2
```

shift_right:

```
addi s5, s3, 4 # Lấy địa chỉ của A[j+1]
add s6, a0, s5 # Tính địa chỉ trong mảng
sw  a2, 0(s6) # A[j+1] = A[j]
addi s3, s3, -4 # j--
j   Loop2
```

endLoop2:

```
addi s5, s3, 4  # Lấy địa chỉ của A[j+1]

add s6, a0, s5  # Tính địa chỉ trong mảng

sw a1, 0(s6)  # A[j+1] = key

addi s0, s0, 4  # i++

j Loop1
```

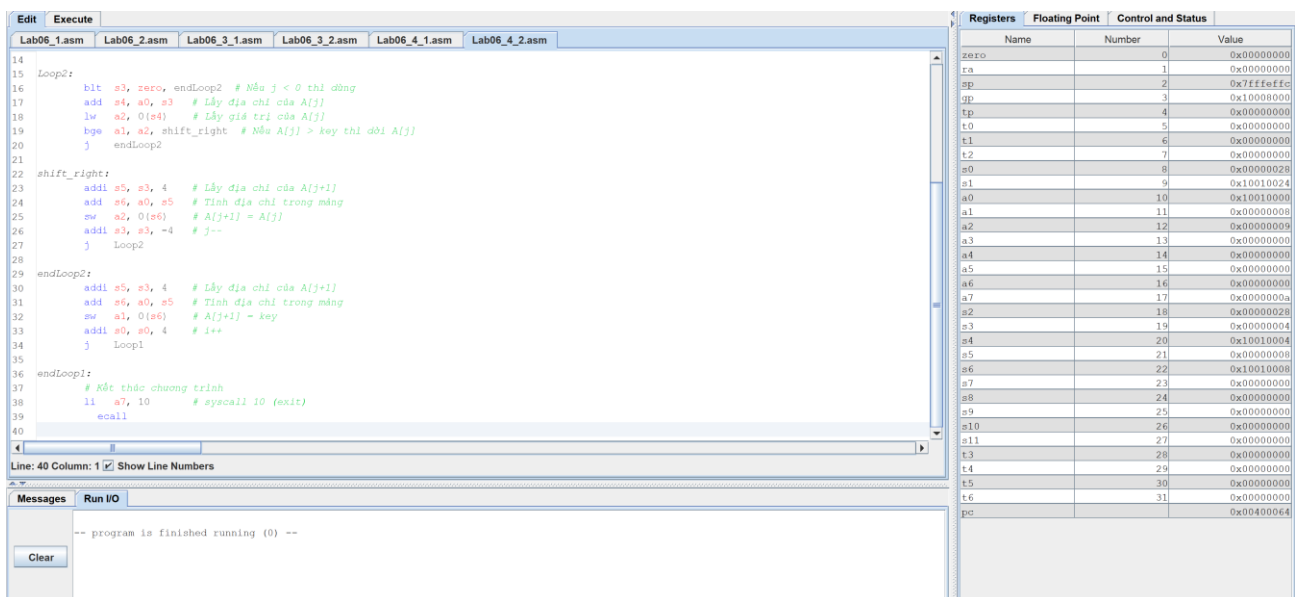
endLoop1:

```
# Kết thúc chương trình

li a7, 10  # syscall 10 (exit)

ecall
```

- **Các bước thực hiện của chương trình:**
 - + Tương tự sắp xếp tăng dần nhưng thay đổi điều kiện để dịch phải
- **Chạy chương trình:**



- **Kết quả của chương trình:**
 - + Với giá trị của mảng A là : 1, 3, 5, 7, 9, 10, 2, 4, 6, 8
 - + Giá trị ban đầu của cửa sổ Data segment là:

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000001	0x00000003	0x00000005	0x00000007	0x00000009	0x0000000a	0x00000002	0x00000004
0x10010020	0x00000006	0x00000008	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

+ Giá trị của của số Data segment sau khi thực hiện chương trình là:

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007	0x00000008
0x10010020	0x00000009	0x0000000a	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

⇒ Chương trình hoạt động bình thường với độ phức tạp $O(n^2)$