BÁO CÁO THỨC HÀNH KIẾN TRÚC MÁY TÍNH TUẦN 6

Họ và tên: Phan Khánh Vũ

MSSV: 20235880

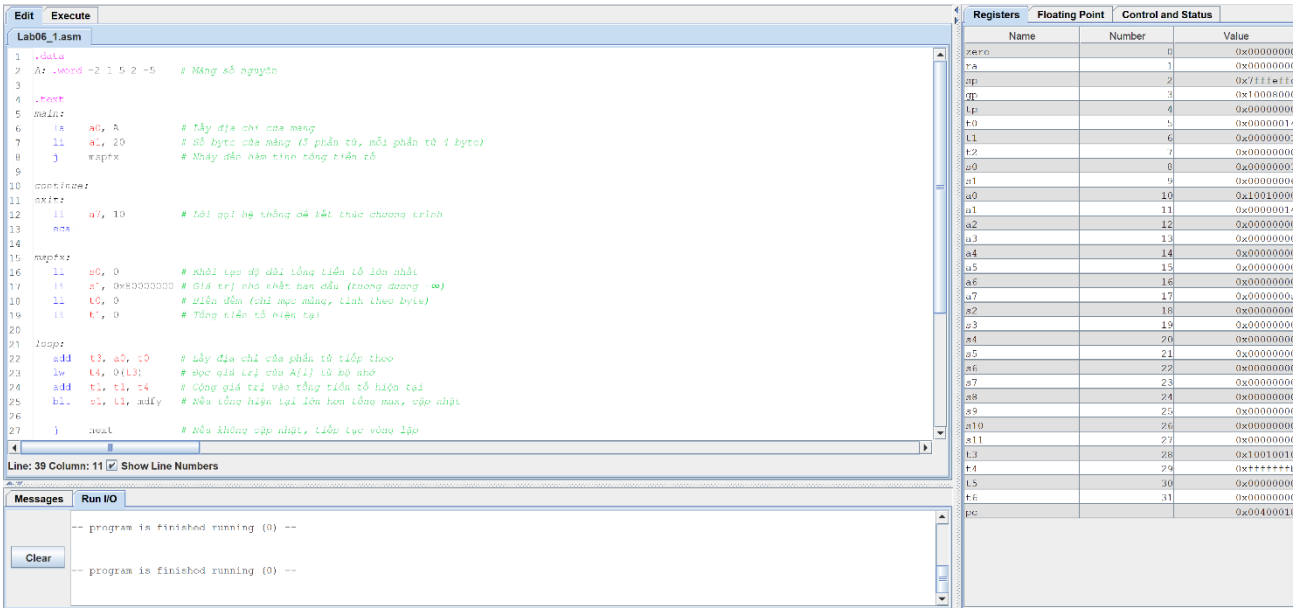
1. **Assignment 1:**

Nhập chương trình:

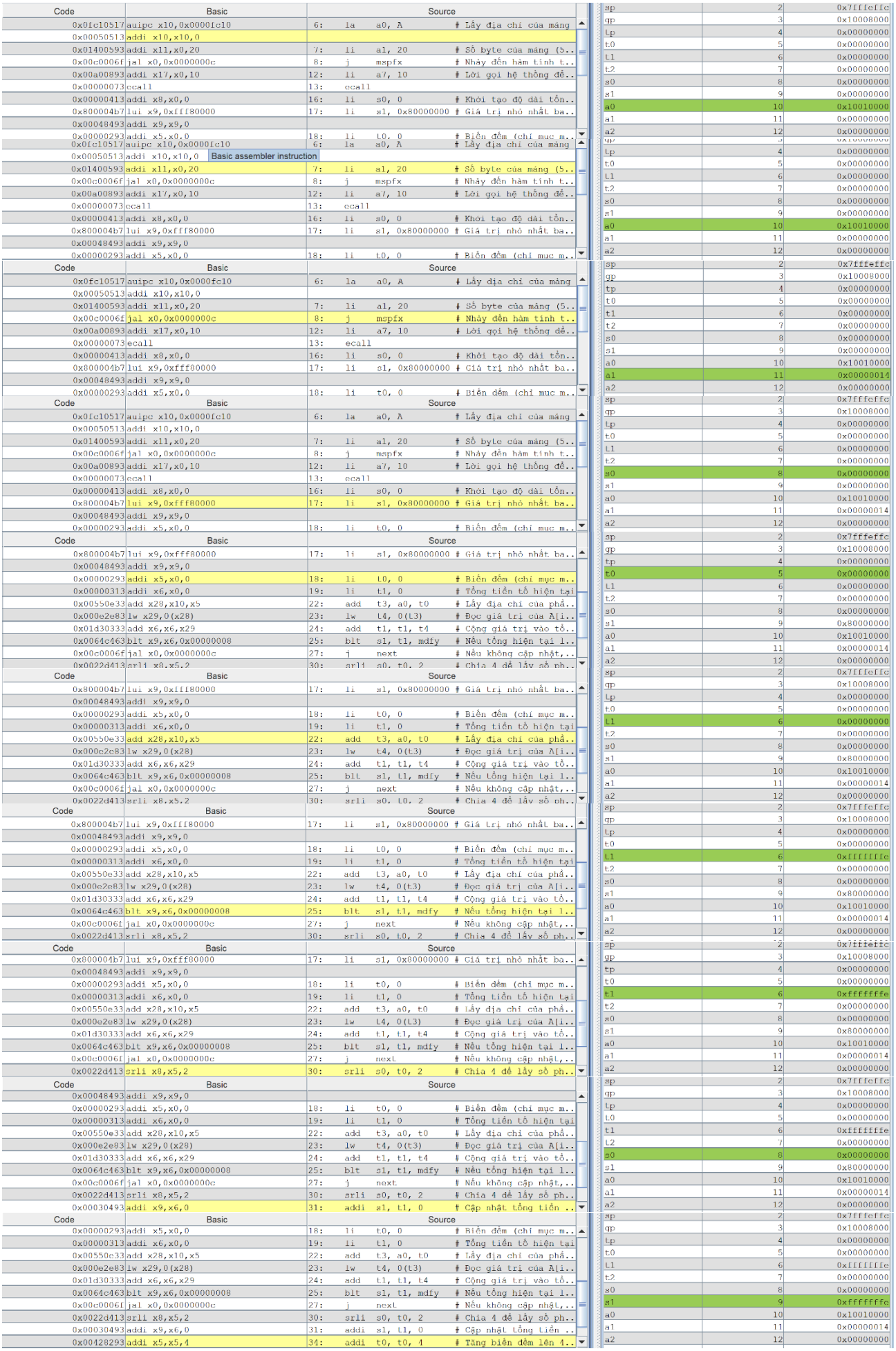
```
.data
A: .word -2 1 5 2 -5    # Mảng số nguyên
.text
main:
  la   a0, A          # Lấy địa chỉ của mảng
  li   a1, 20         # Số byte của mảng (5 phần tử, mỗi phần tử 4 byte)
  j    mspfx          # Nhảy đến hàm tính tổng tiền tố
continue:
exit:
  li   a7, 10         # Lời gọi hệ thống để kết thúc 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
mspfx_end:
```

- **Các bước thực hiện của chương trình:**
  + Cụm .data để khởi tạo giá trị cho mảng
  + Cụm 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.
  + Cụm exit sau khi đã thực hiện xong hàm tính tổng tiền tố chương trình quay lại đây và kết thúc chương trình
  + Cụm câu lệnh li để khởi tạo các giá trị sắp được sử dụng
  + Cụm 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 countinue và kết thúc chương trình
- **Chương trình:**



- **Quan sát sự thay đổi của thanh ghi:**


Code Basic Source
0x0fc10517 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 # Lời gọi hệ thống để..
0x00000073 ecall 13: ecall
0x00000413 addi x8,x0,0 16: li s0, 0 # Khởi tạo độ dài tổn..
0x800004b7 lui x9,0xfff80000 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 tố hiện tại
0x00550e33 add x28,x10,x5 22: add t3, a0, t0 # Lấy địa chỉ của phầ..
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,x6,0x00000008 25: blt s1, t1, mdfy # Nếu tổng hiện tại l..
0x00c0006f jal x0,0x0000000c 27: j next # Nếu không cập nhật,..
0x0022d413 srli x8,x5,2 30: srli 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..
Basic assembler instruction
sp 2 0x7fffeffc
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

- Sau đó vòng lặp sẽ tiếp tục, theo lí 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	0x7fffeffc
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	0xfffffffe	0x00000001	0x00000005	0x00000002	0xfffffffb
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

  ⇨ Chương trình chạy đúng với thuật toán.

2. Assignment 2:

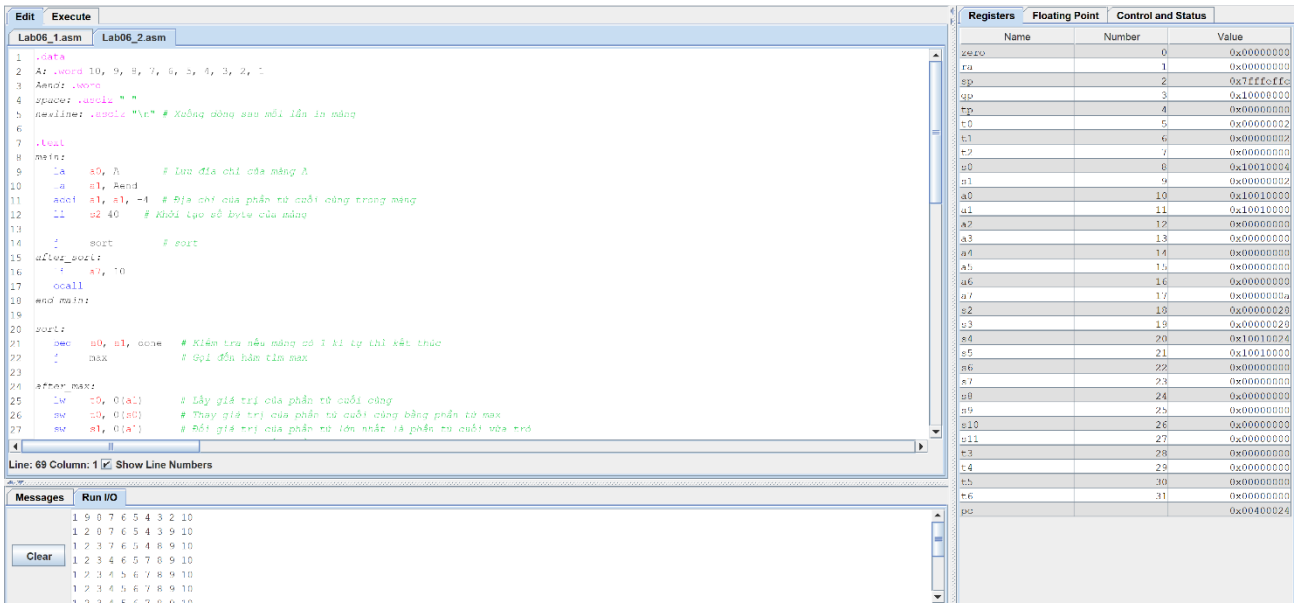
Nhập chương trình:

```
.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:
```

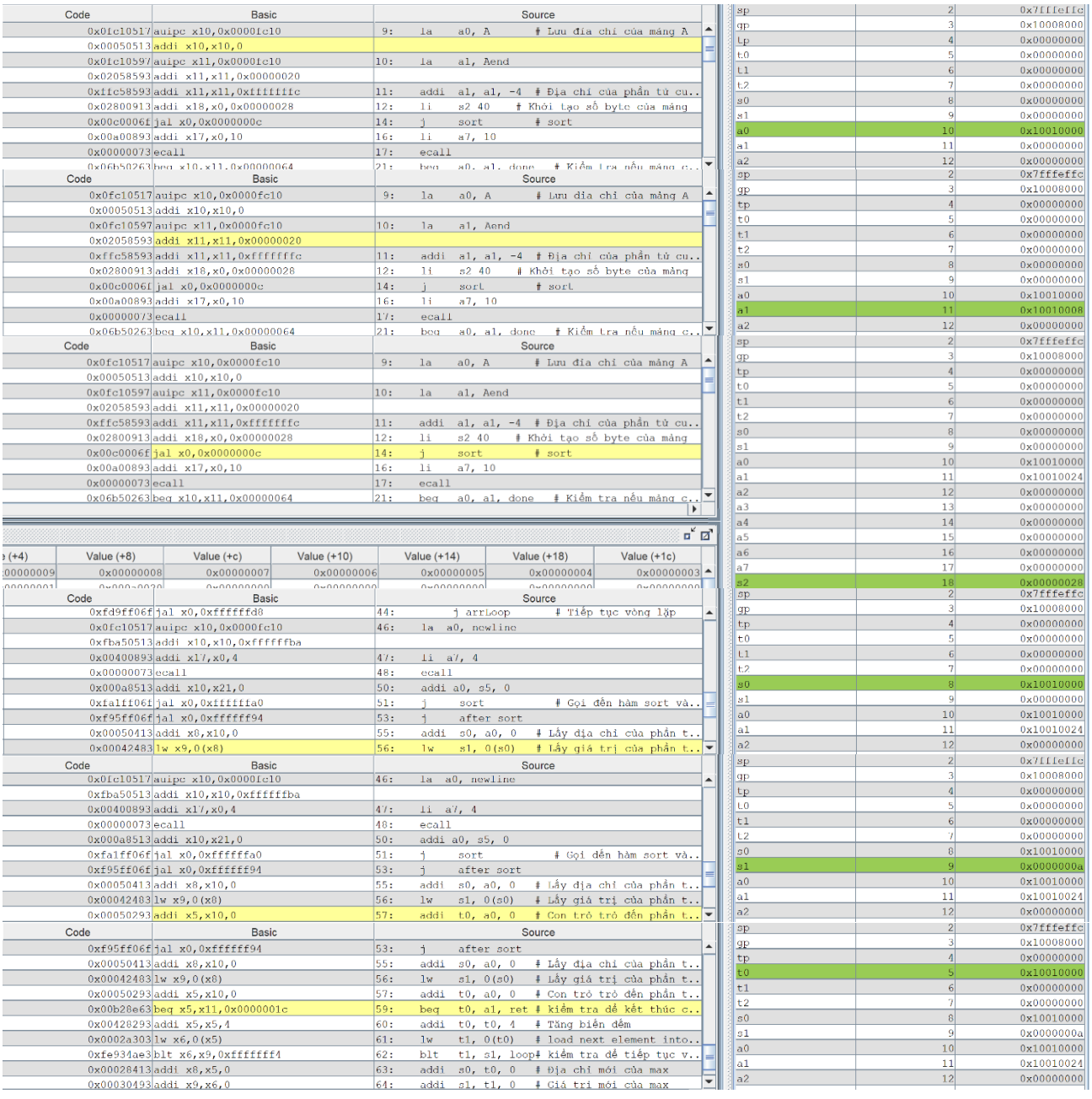
```
  la    a0, A       # Lưu đỉa chỉ của mảng A
  la    a1, Aend
  addi  a1, a1, -4  # Địa chỉ của phần tử cuối cùng trong mảng
  li    s2 40    # Khởi tạo số byte của mảng

  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 trỏ
  addi  a1, a1, -4     # Lùi còn trỏ 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
```

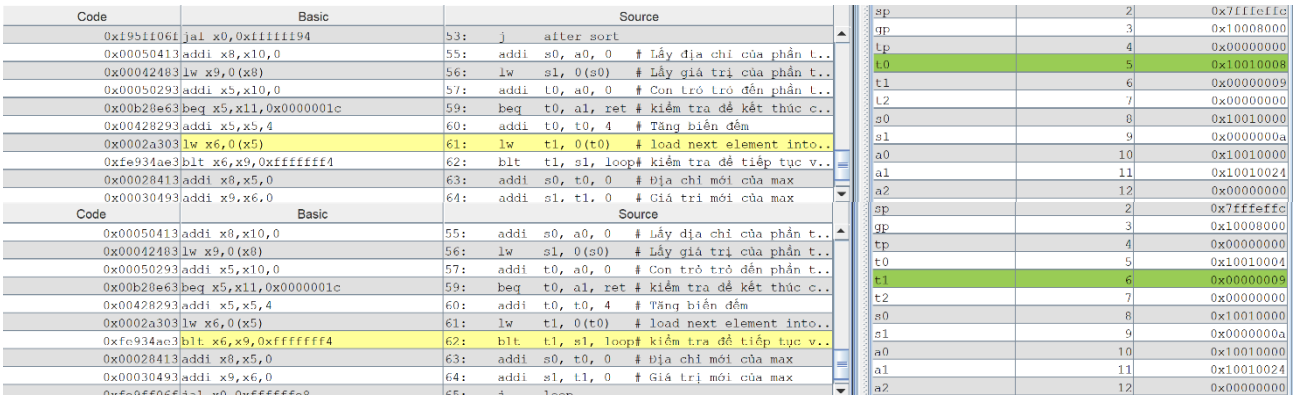
```
        li  a7, 4
        ecall

        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 để 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:**



- **Quan sát sự thay đổi của thanh ghi:**





Code	Basic	Source
0x1951f06f	jal x0,0xfffff194	53: j after_sort
0x00050413	addi x8,x10,0	55: addi s0, a0, 0 # Lấy địa chỉ của phần t..
0x00042483	lw x9,0(x8)	56: lw s1, 0(s0) # Lấy giá trị của phần t..
0x00050293	addi x5,x10,0	57: addi t0, a0, 0 # Con trỏ trỏ đến phần t..
0x00b28e63	beq x5,x11,0x0000001c	59: beq t0, a1, ret # kiểm tra để kết thúc c..
0x00428293	addi x5,x5,4	60: addi t0, t0, 4 # Tăng biến đếm
0x0002a303	lw x6,0(x5)	61: lw t1, 0(t0) # load next element into..
0xfe934ae3	blt x6,x9,0xfffffff4	62: blt t1, s1, loop# kiểm tra để tiếp tục v..
0x00028413	addi x8,x5,0	63: addi s0, t0, 0 # Địa chỉ mới của max
0x00030493	addi x9,x6,0	64: addi s1, t1, 0 # Giá trị mới của max

Code	Basic	Source
0x00050413	addi x8,x10,0	55: addi s0, a0, 0 # Lấy địa chỉ của phần t..
0x00042483	lw x9,0(x8)	56: lw s1, 0(s0) # Lấy giá trị của phần t..
0x00050293	addi x5,x10,0	57: addi t0, a0, 0 # Con trỏ trỏ đến phần t..
0x00b28e63	beq x5,x11,0x0000001c	59: beq t0, a1, ret # kiểm tra để kết thúc c..
0x00428293	addi x5,x5,4	60: addi t0, t0, 4 # Tăng biến đếm
0x0002a303	lw x6,0(x5)	61: lw t1, 0(t0) # load next element into..
0xfe934ae3	blt x6,x9,0xfffffff4	62: blt t1, s1, loop# kiểm tra để tiếp tục v..
0x00028413	addi x8,x5,0	63: addi s0, t0, 0 # Địa chỉ mới của max
0x00030493	addi x9,x6,0	64: addi s1, t1, 0 # Giá trị mới của max

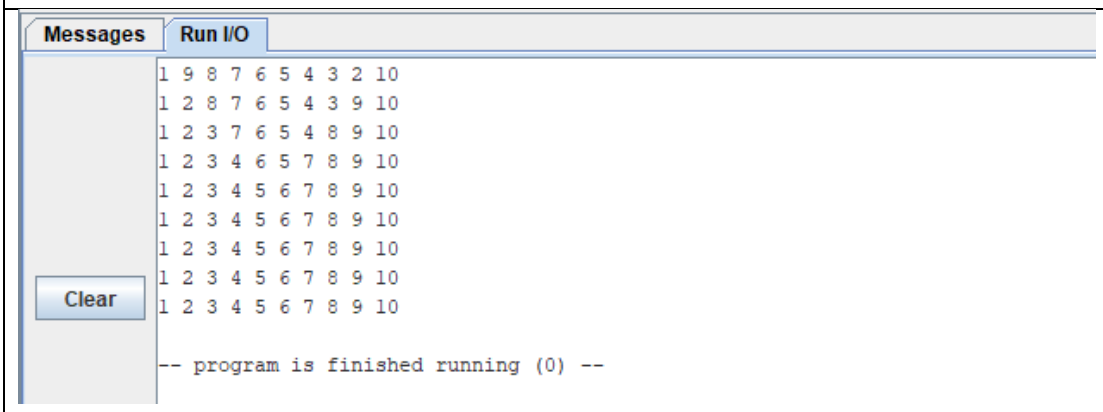
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x10010008
t1	6	0x00000009
t2	7	0x00000000
s0	8	0x10010000
s1	9	0x0000000a
a0	10	0x10010000
a1	11	0x10010024
a2	12	0x00000000

sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x10010004
t1	6	0x00000009
t2	7	0x00000000
s0	8	0x10010000
s1	9	0x0000000a
a0	10	0x10010000
a1	11	0x10010024
a2	12	0x00000000

- Chương trình sẽ tiếp tục vòng lặp t1 nhận các giá trị giảm dần 8 -7-6-5-4-3-2-1. Với giá trị của mảng A được khởi tạo là 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 kết quả theo thuật toán sẽ là: 1 2 3 4 5 6 7 8 9 10

- **Kiểm tra kết quả thực hiện của chương trình:**
  + Với giá trị của mảng A được khởi tạo là 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

Kết quả ở cửa sổ Run I/O là:



Giá trị của các thanh ghi ở cửa sổ Registers là:

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
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	0x000a0020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

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
```

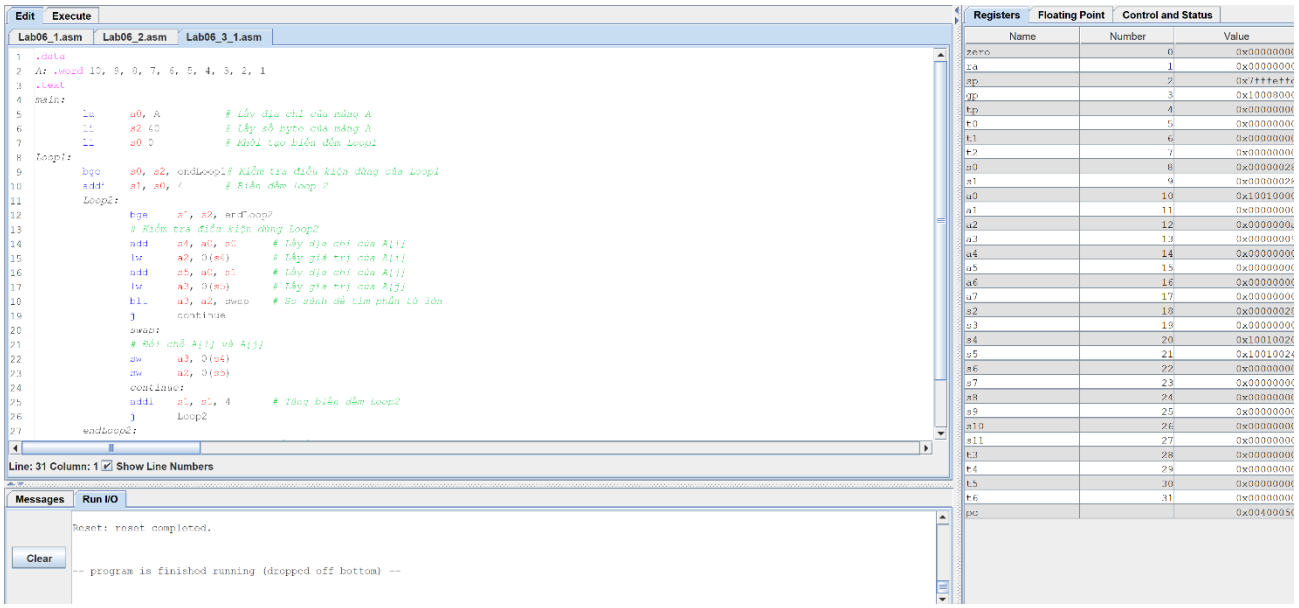
```
Loop1:
        bge    s0, s2, endLoop1# Kiểm tra điều kiện dừng của Loop1
        addi   s1, s0, 4       # Biến đếm loop 2
        Loop2:
                bge    s1, s2, endLoop2
                # Kiểm tra điều kiện dừng Loop2
                add    s4, a0, s0     # Lấy địa chỉ của A[i]
                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 gía 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 0(n^2)

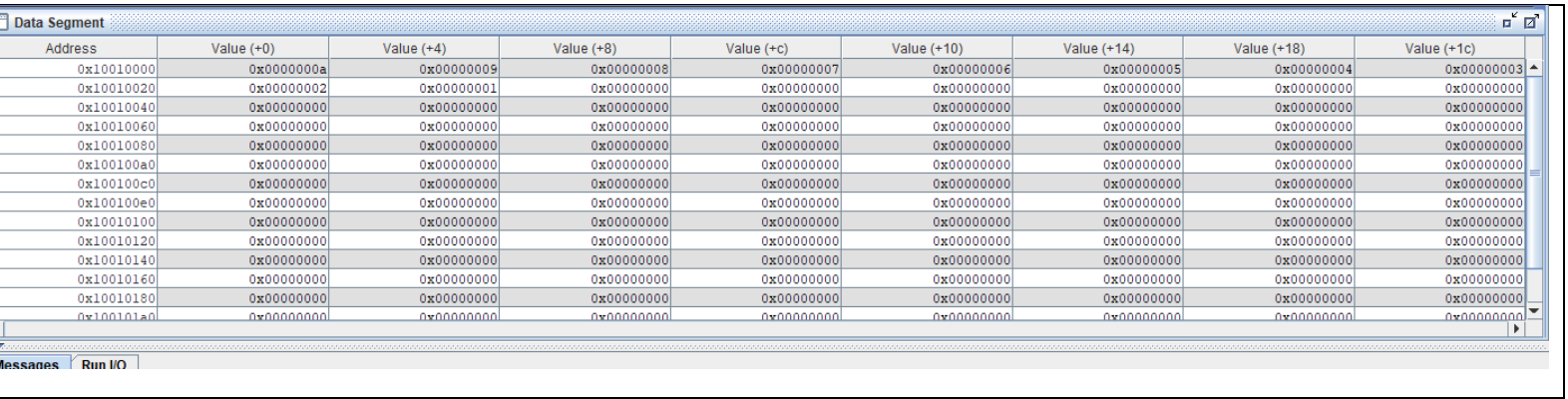
- **Chạy chương trình:**



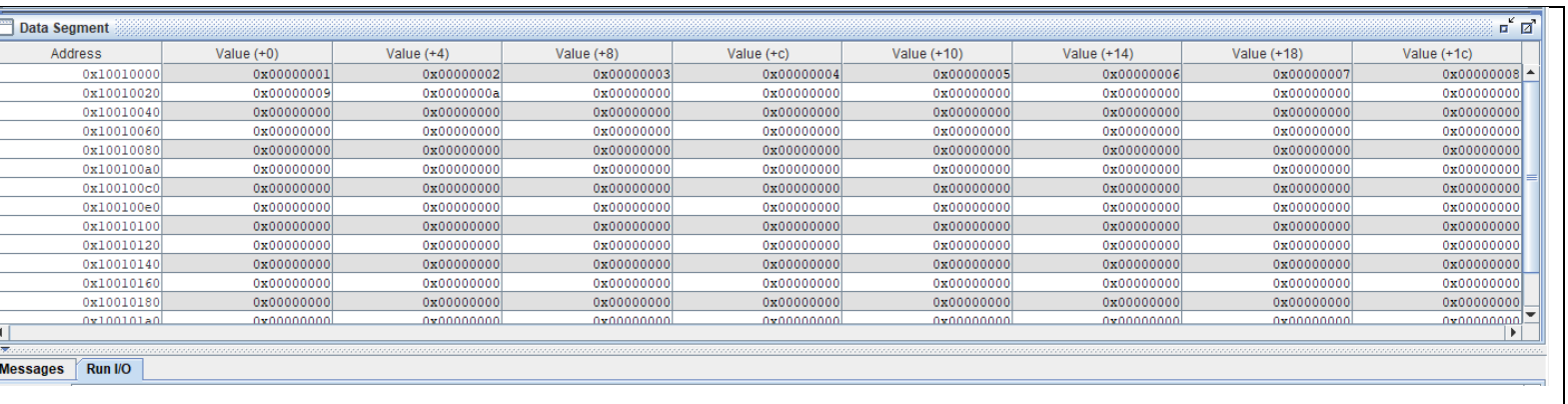
- **Kết quả thực hiện chương trình:**

+ 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



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



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

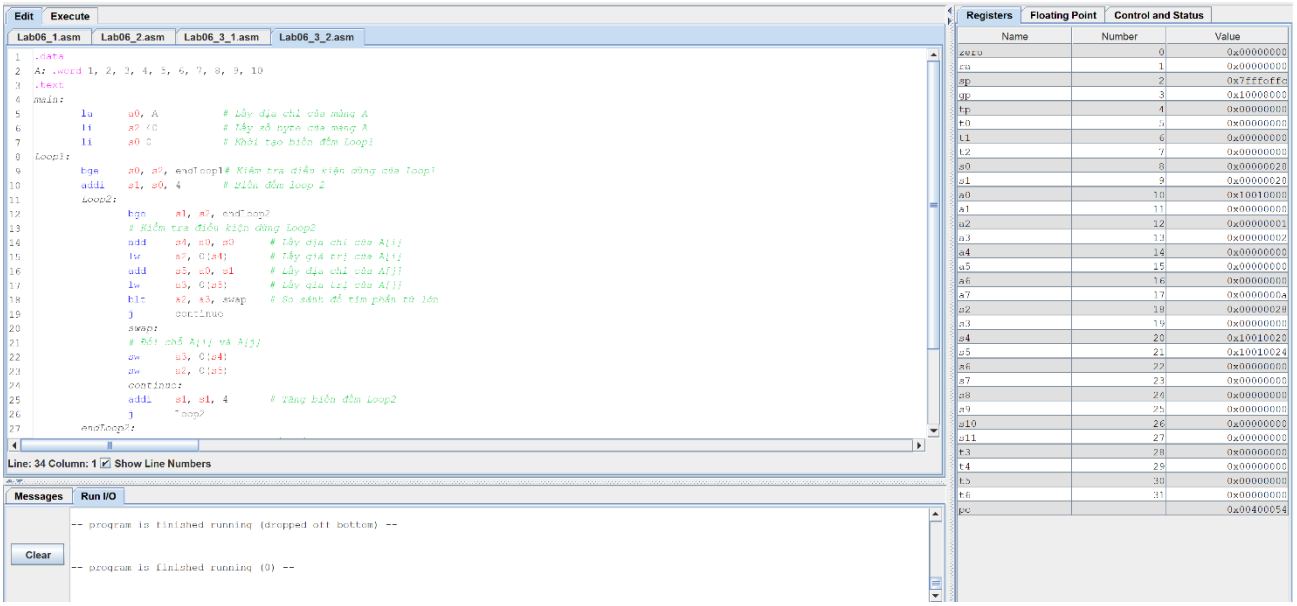
```
.data
        A: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
.text
main:
        la      a0, A           # Lấy địa chỉ của mảng A
        li      s2 40           # Lấy số byte của mảng A
        li      s0 0            # Khởi tạo biến đếm Loop1
Loop1:
        bge     s0, s2, endLoop1# Kiểm tra điều kiện 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 gía 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 kiệm để swap A[i] và A[j] thay đổi so với chương trình sắp xếp tăng dần.

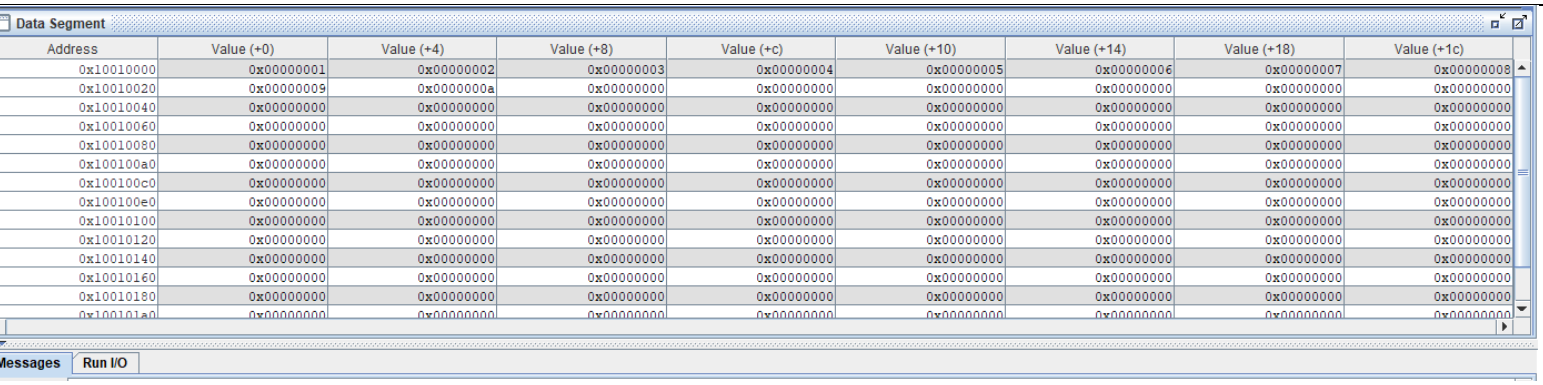
- **Chạy chương trình:**



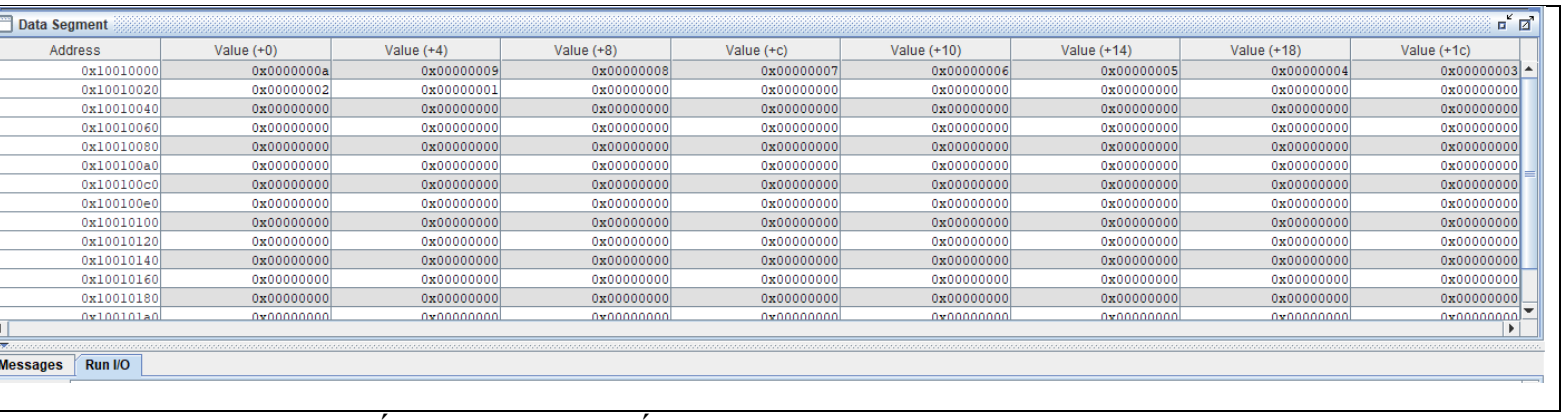
- **Kết quả của chương trình:**

  + 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:



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



⇨ Kết quả đúng với kết quả tính toán.

4. Assignment 4 (insertion sort)

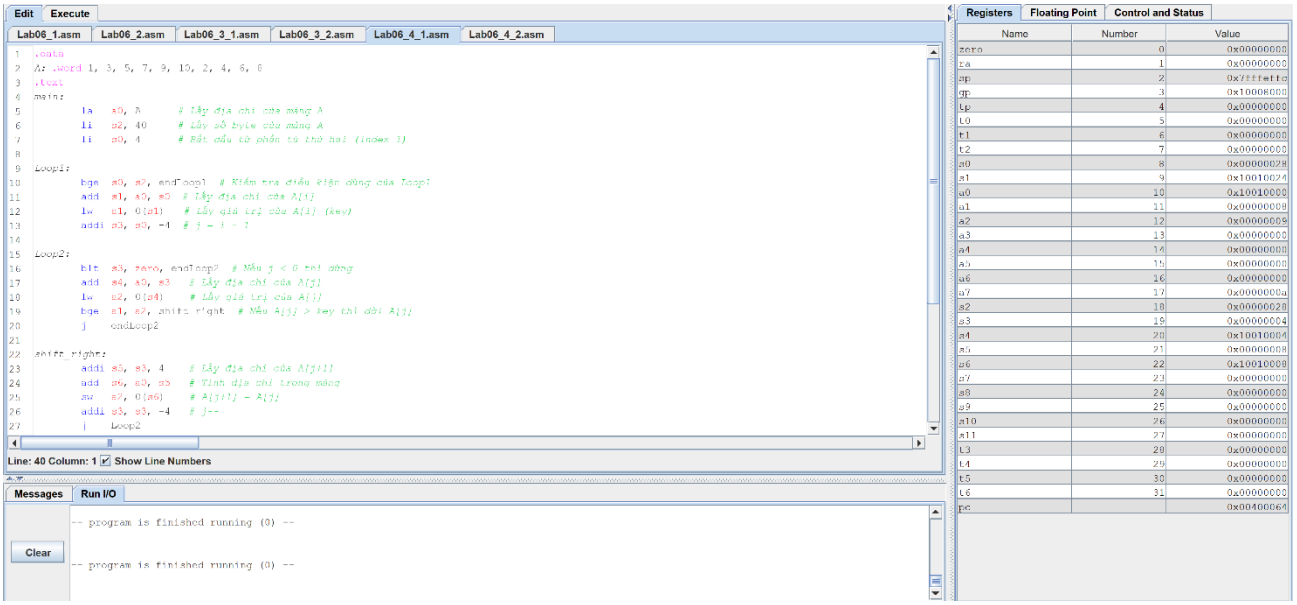
- **Nhập chương trình: (sắp xếp tăng dần)**

```
.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:**
  + Cụm .data để khai báo giá trị của các phần tử trong mảng
  + Cụm 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.

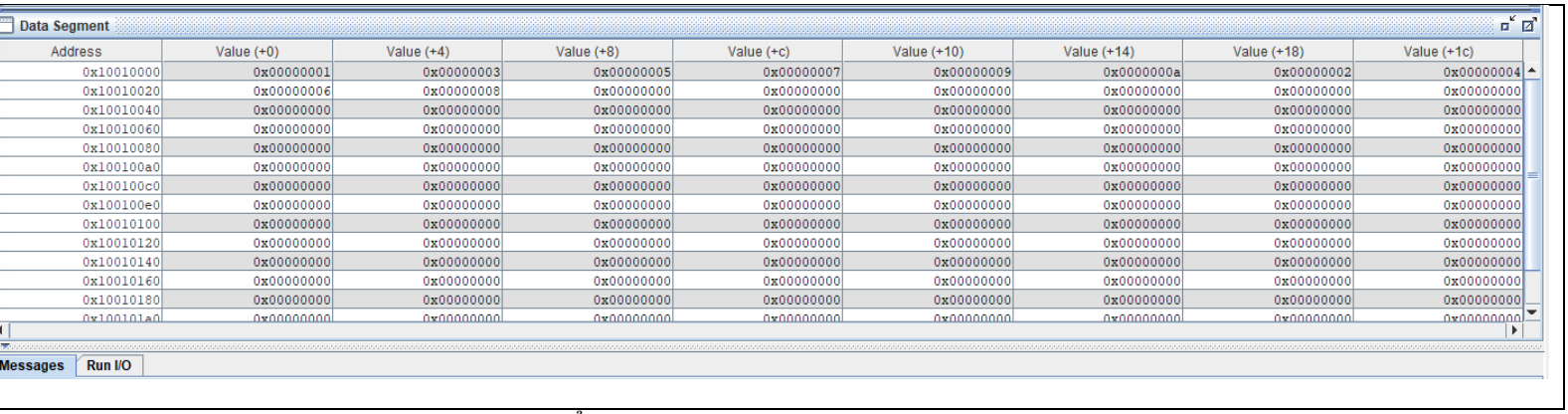
- **Chạy chương trình:**



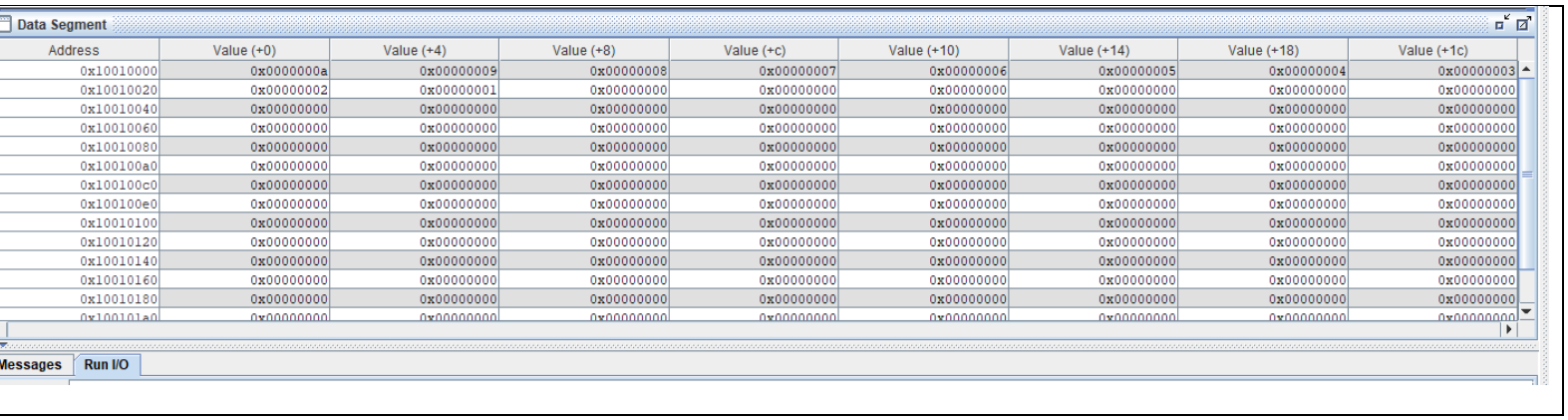
- **Kết quả của chương trình**

+ 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:



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

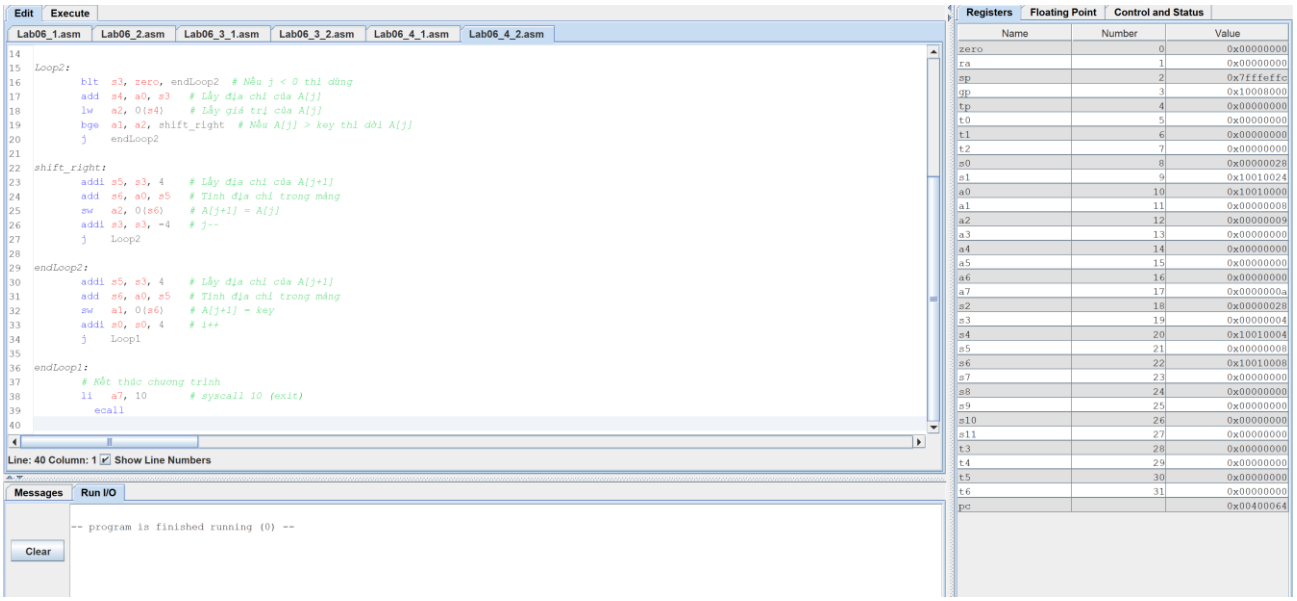


- **Nhập chương trình: (Sắp xếp giảm dần)**

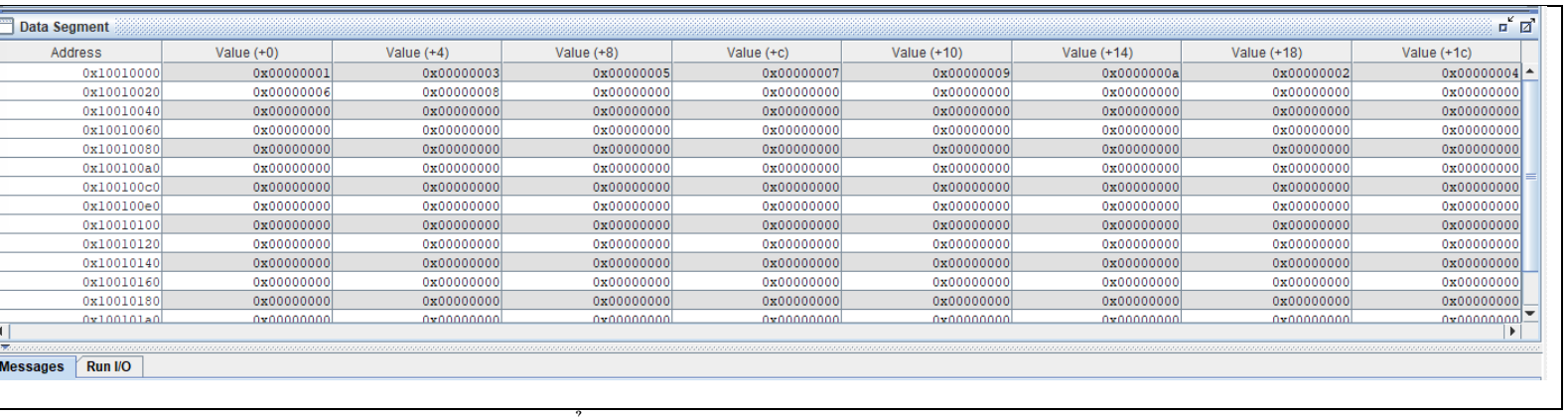
```
.data
A: .word 1, 3, 5, 7, 9, 10, 2, 4, 6, 8
.text
main:
        la   a0, A      # Lấy địa chỉ của mảng A
        li   s2, 40     # Lấy số byte của mảng A
        li   s0, 4      # Bắt đầu từ phần tử thứ hai (index 1)

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
```

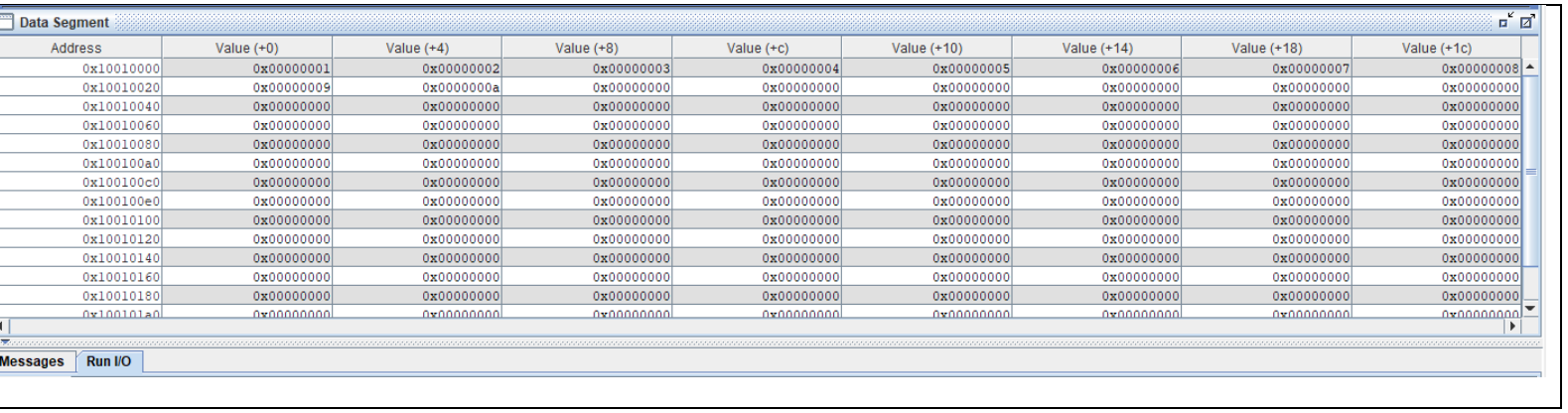
- **Các bước thực hiện của chương trình:**
  + Tương tự săp xếp tăng dần nhưng thay đổi điều kiện để dịch phải
- **Chạy chương trình:**



- **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:



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



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