

Bài 6. Mảng và con trỏ

Họ và tên: Nguyễn Thành Duy

MSSV: 20235696

Assignment 1

Code:

```
.data
A: .word -1, 6, 1, -2, 0
.text
main:
la a0, A
li a1, 5
j mspfx
continue:
exit:
li a7, 10
ecall
end_of_main:
mspfx:
li s0, 0 # initialize length of prefix-sum in s0 to 0
li s1, 0x80000000 # initialize max prefix-sum in s1 to smallest int
li t0, 0 # initialize index for loop i in t0 to 0
li t1, 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
blt s1, t1, mdfy # if (s1 < t1) modify results
j next
mdfy:
addi s0, t0, 1 # new max-sum prefix has length i+1
addi s1, t1, 0 # new max sum is the running sum
next:
addi t0, t0, 1 # advance the index i
blt t0, a1, loop # if (i<n) repeat
```

```
done:
j continue
mspfx_end:
```

Sau lần lặp thứ nhất: Ứng với độ dài 1 thì $s1 = -1$

s1	9	0xffffffff
----	---	------------

Sau lần lặp thứ hai: Ứng với độ dài 2 thì $s1 = -1 + 6 = 5$

s1	9	0x00000005
----	---	------------

Cứ lặp tiếp thì kết thúc chương trình: Thanh ghi $s1 = 6$ (tổng max của các phần tử liên tiếp)

s1	9	0x00000006
----	---	------------

Assignment 2

Code:

```
.data
A: .word 7, -2, 5, 1, 5, 6, 7, 3, 6, 8, 8, 59, 5
Aend: .word
line: .asciz "\n"
space: .ascii " "
.text
la a3, A
la a1, Aend
addi a1, a1, -4
li t2, 13 # length of A
j sort
done:
li a7, 10
ecall

sort:
beq a3, a1, done
j max

after_max:
lw t0, 0(a1) # load last element into t0
sw s1, 0(a1) # copy max value to last element
sw t0, 0(s0) # copy last element to max location
addi a1, a1, -4
print:
la s2, A
li t0, 0
```

```

# print array A after_max
print_loop:
add t3, t0, t0
add t3, t3, t3
add s3, t3, s2
li a7, 1
lw a0, 0(s3)
ecall
addi t0, t0, 1
bge t0, t2, end_print_loop
# print space
li a7, 4
la a0, space
ecall
j print_loop
end_print_loop:
# print new line
li a7, 4
la a0, line
ecall

```

```

j sort

```

```

max:
addi s0, a3, 0
lw s1, 0(s0)
addi t0, a3, 0
loop:
beq t0, a1, ret
addi t0, t0, 4
lw t1, 0(t0)
blt t1, s1, loop
addi s0, t0, 0
addi s1, t1, 0
j loop
ret:
j after_max

```

Giải thích: a0, a1 là con trỏ, trỏ đến đầu và cuối mảng A. Ở mỗi lần lặp lần lượt tìm phần tử lớn nhất trong mảng chèn vào cuối rồi dịch chuyển con trỏ a1 về trước cứ như vậy tới khi nào a0 = a1 thì chương trình kết thúc.

Kết quả sau khi chạy:

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x0f10697	auipc x13,0x0000fc10	7: la a3, A
	0x00400004	0x0006693	addi x13,x13,0	
	0x00400008	0x0f10597	auipc x11,0x0000fc10	8: la a1, Aend
	0x0040000c	0x02c58593	addi x11,x11,0x0000002c	
	0x00400010	0xffc58593	addi x11,x11,0xfffffff0	9: addi a1, a1, -4
	0x00400014	0x00400393	addi x7,x0,13	10: li t2, 13 # length of A
	0x00400018	0x00b006ef	jal x0,0x0000000c	11: j sort
	0x0040001c	0x00a00859	addi x17,x0,10	13: li a7, 10
	0x00400020	0x00000073	ecall	14: ecall
	0x00400024	0x0feb68ce3	beq x13,x11,0xfffffff0	18: beq a3, a1, done
	0x00400028	0x0e80006ef	jal x0,0x00000048	19: j max
	0x0040002c	0x0005a323	le w6,0(x11)	22: lw a0, 0(x11) # load last element into

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0xffffffff	0x00000001	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000007	0x00000007	0x00000008	0x00000008	0x0000000b	0x0020000a	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

Registers

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffefc
fp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000004
t1	6	0xffffffff
t2	7	0x0000000b
a0	8	0x10010000
a1	9	0x00000001
a2	10	0x10010030
a3	11	0x10010000
a4	12	0x00000000
a5	13	0x10010000
a6	14	0x00000000
a7	15	0x00000000
a8	16	0x00000000
a9	17	0x0000000a
a10	18	0x10010000
a11	19	0x10010030
a12	20	0x00000000
a13	21	0x00000000
a14	22	0x00000000
a15	23	0x00000000
a16	24	0x00000000
a17	25	0x00000000
a18	26	0x00000000
a19	27	0x00000000
a20	28	0x00000030
a21	29	0x00000000
a22	30	0x00000000
a23	31	0x00000000
pc		0x00400024

Messages

```

7 -2 5 1 5 6 7 3 6 8 8 59
7 -2 5 1 5 6 7 3 6 8 8 59
7 -2 5 1 5 6 7 3 6 8 8 59
7 -2 5 1 5 6 5 3 6 7 8 8 59
6 -2 5 1 5 6 5 3 7 7 8 8 59
6 -2 5 1 3 3 5 6 7 7 8 8 59
5 -2 5 1 3 6 6 7 7 8 8 59

```

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x0f10697	auipc x13,0x0000fc10	7: la a3, A
	0x00400004	0x0006693	addi x13,x13,0	
	0x00400008	0x0f10597	auipc x11,0x0000fc10	8: la a1, Aend
	0x0040000c	0x02c58593	addi x11,x11,0x0000002c	
	0x00400010	0xffc58593	addi x11,x11,0xfffffff0	9: addi a1, a1, -4
	0x00400014	0x00400393	addi x7,x0,13	10: li t2, 13 # length of A
	0x00400018	0x00b006ef	jal x0,0x0000000c	11: j sort
	0x0040001c	0x00a00859	addi x17,x0,10	13: li a7, 10
	0x00400020	0x00000073	ecall	14: ecall
	0x00400024	0x0feb68ce3	beq x13,x11,0xfffffff0	18: beq a3, a1, done
	0x00400028	0x0e80006ef	jal x0,0x00000048	19: j max
	0x0040002c	0x0005a323	le w6,0(x11)	22: lw a0, 0(x11) # load last element into

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
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
0x100101c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Registers

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffefc
fp	3	0x10008000
tp	4	0x00000000
t0	5	0x0000000d
t1	6	0xffffffff
t2	7	0x0000000b
a0	8	0x10010000
a1	9	0x00000001
a2	10	0x10010030
a3	11	0x10010000
a4	12	0x00000000
a5	13	0x10010000
a6	14	0x00000000
a7	15	0x00000000
a8	16	0x00000000
a9	17	0x0000000a
a10	18	0x10010000
a11	19	0x10010030
a12	20	0x00000000
a13	21	0x00000000
a14	22	0x00000000
a15	23	0x00000000
a16	24	0x00000000
a17	25	0x00000000
a18	26	0x00000000
a19	27	0x00000000
a20	28	0x00000030
a21	29	0x00000000
a22	30	0x00000000
a23	31	0x00000000
pc		0x00400024

Messages

```

5 -2 5 1 5 6 7 7 8 8 59
5 -2 5 1 3 5 6 7 7 8 8 59
5 -2 3 1 5 5 6 7 7 8 8 59
1 -2 3 5 5 6 7 7 8 8 59
1 -2 3 5 5 3 6 7 7 8 8 59
2 1 3 5 5 6 7 7 8 8 59

```

Mảng ban đầu: 7, -2, 5, 1, 5, 6, 7, 3, 6, 8, 8, 59, 5

Mảng sau mỗi lượt sắp xếp: -2 1 3 5 5 5 6 6 7 7 8 8 59

Assignment 3

Code:

```

.data
A: .word 5, 1, 4, -2, 7, 5, 6, 9, 6
Aend: .word
newline: .asciz "\n"
.text
main:
la t6, A # a0 = địa chỉ của A[0]
la a1, Aend
addi a1, a1, -4 # a1 = địa chỉ của A[n-1]
j bubble_sort # gọi thủ tục sắp xếp
after_sort:

```

```

    li    a7, 10
    ecall
end_main:
# -----
# Thủ tục Bubble Sort (Sắp xếp nổi bọt)
# Cách sử dụng thanh ghi:
# t6 - con trỏ đến mảng A
# a1 - con trỏ đến phần tử cuối cùng
# t1 - con trỏ đến phần tử đang xét
# t2 - giá trị A[i]
# t3 - giá trị A[i+1]
# -----
bubble_sort:
outer_loop:
    beq   t6, a1, done    # nếu chỉ còn 1 phần tử thì done
    la    t1, A
inner_loop:
    addi  t2, t1, 4        # t2 chứa địa chỉ của A[i+1]
    bge   t1, a1, next_outer # nếu đến cuối của mảng thì kết thúc inner_loop
    lw    t3, 0(t1)        # A[i]
    lw    t4, 0(t2)        # A[i+1]
    ble   t3, t4, no_swap  # kiểm tra xem cần đổi chỗ không
    sw    t3, 0(t2)
    sw    t4, 0(t1)        # thực hiện đổi chỗ
no_swap:
    addi  t1, t1, 4        # trở đến phần tử tiếp theo
    j     inner_loop       # tiếp tục vòng lặp trong
next_outer:
    addi  a1, a1, -4       # cố định phần tử cuối dãy hiện tại
print:
    la    a5, A            # lấy lại địa chỉ đầu mảng A
    la    a6, Aend
    addi  a6, a6, -4       # lấy địa chỉ cuối mảng A
    j     print_array
continue:
    j     outer_loop       # bắt đầu sắp xếp như cũ nhưng với mảng nhỏ hơn 1 phần tử
done:
    j     after_sort
# chương trình in mảng
print_array:
    bgt   a5, a6, print_end # nếu con trỏ đến phần tử cuối cùng dừng việc in
    li    a7, 1
    lw    a0, 0(a5)        # lấy giá trị A[i] hiện tại
    ecall                                # in ra
    li    a7, 11
    li    a0, 32           # Mã ASCII của dấu cách
    ecall                                # in khoảng trắng giữa 2 số liên tiếp
    addi  a5, a5, 4
    j     print_array
print_end:
    li    a7, 4

```

```

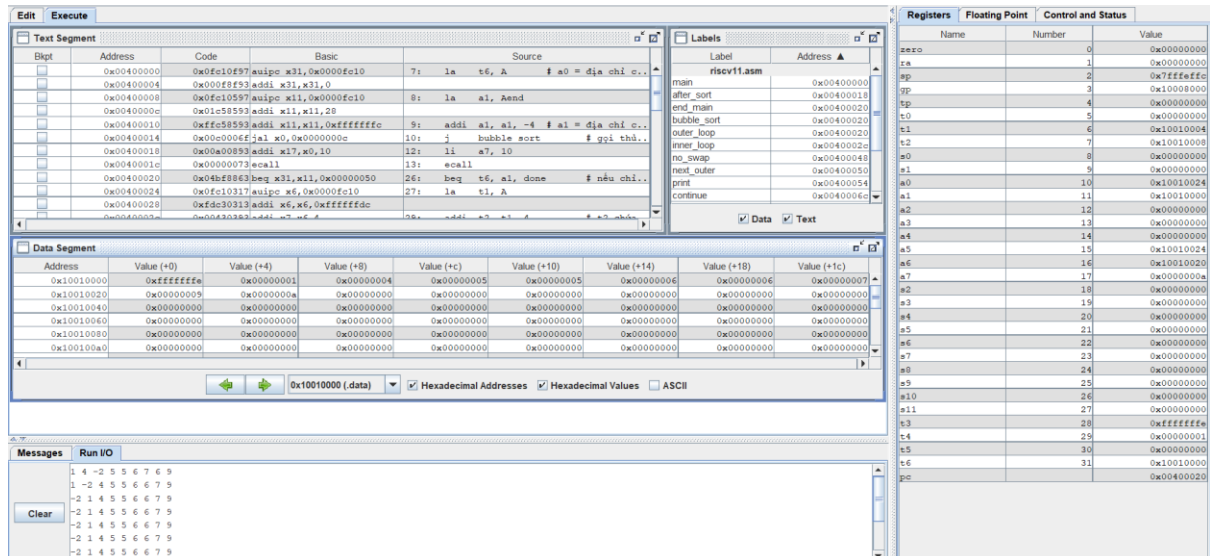
la a0, newline
ecall          # in dấu xuống dòng
j continue

```

Kết quả:

Mảng ban đầu: 5, 1, 4, -2, 7, 5, 6, 9, 6

Mảng sau khi sắp xếp: -2, 1, 4, 5, 5, 6, 6, 7, 9



Giải thích: Thuật toán lặp lại quá trình so sánh $a[i]$ và $a[i+1]$ nếu như $a[i] > a[i+1]$ thì swap còn không dịch chuyển con trỏ để so sánh 2 phần tử tiếp theo. Kết thúc 1 vòng lặp thì ta sẽ chuyển được giá trị lớn nhất xuống cuối mảng và cứ tiếp tục như vậy khi con trỏ đầu = con trỏ cuối.

Assignment 4

Code:

```

.data
A: .word 5, 1, 4, -2, 7, 5, 6, 9, 6
Aend: .word
dau_cach: .asciz " "
newline: .asciz "\n"
.text
main:
    la t6, A      # t6 = địa chỉ đầu của A
    la a1, Aend
    addi a1, a1, -4 # a1 = địa chỉ của A[n-1]
    j insertion_sort # Gọi thủ tục sắp xếp

```

```

after_sort:
    li  a7, 10
    ecall
end_main:
# -----
# Thủ tục Insertion Sort (Sắp xếp chèn)
# -----
insertion_sort:
    la  t0, A
    addi t0, t0, 4 # lấy địa chỉ phần tử thứ 2 của mảng
outer_loop:      # Duyệt từ A[1] đến A[n-1] mỗi lần chọn A[i] làm key
    bgt  t0, a1, done # nếu t1 vượt quá địa chỉ cuối mảng thì sắp xếp xong
    lw  t1, 0(t0)    # A[i]
    addi t2, t0, -4  # lấy địa chỉ phần tử trước đó t2 = j = i-1

inner_loop:      # tìm vị trí thích hợp để chèn key
    blt  t2, t6, insert_key # nếu j < 0, thì chèn key

    lw  t3, 0(t2)    # Lấy A[i-1]
    bgt  t3, t1, shift_right # nếu A[j] > key (A[i-1] > A[i]) thì dịch A[j] sang phải
    j  insert_key    # nếu không thì chèn key
shift_right:
    sw  t3, 4(t2)    # A[j+1] = A[j], dịch A[j] sang phải
    addi t2, t2, -4  # j = j - 1, dịch sang trái để lấy phần tử tiếp theo cần so
sánh
    j  inner_loop    # Tiếp tục vòng lặp
insert_key:
    sw  t1, 4(t2)    # A[i] = A[j+1] = key (chèn key vào vị trí đúng)
    addi t0, t0, 4    # Lấy địa chỉ phần tử tiếp theo của mảng
print:
    la  a5, A        # lấy lại địa chỉ đầu mảng A
    la  a6, Aend
    addi a6, a6, -4  # lấy địa chỉ cuối mảng A
    j  print_array   # in mảng sau mỗi vòng lặp
continue:
    j  outer_loop    # sau khi in thì xét vòng lặp mới
done:
    j  after_sort
# chương trình in mảng
print_array:

```

Vòng lặp 2: Phần tử thứ 3 là 4 được lấy ra, so sánh $1 < 4 < 5$ nên được xếp vào giữa 1 và 5.

Tương tự như vậy đối với các phần tử khác của mảng.