Thực hành kiến trúc máy tính

Báo cáo thực hành

Bài 6. Mảng và con trỏ

Họ Tên	Lê Thành An
MSSV	20235631

ASSIGNMENT 1

ĐOẠN MÃ :

```
.data
A: .word -2, 4, -1, 5, -2, 3, 4, -7
.text
main:
    la    a0, A
    li    a1, 8
    j     mspfx
continue:
exit:
    li    a7, 10
    ecall
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 reachs.
# @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          # 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:
```

Kết quả:

a0	10	0x10010000
a1	11	0x00000008
s0	8	0x00000000
s1	9	0x80000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
t3	28	0x10010000
t4	29	0xfffffffe
t1	6	0xfffffffe
s0	8	0x00000001
s1	9	0xfffffffe
t0	5	0x00000001
t2	7	0x00000002
t2	7	0x00000004
t3	28	0x10010004
t4	29	0x00000004
t1	6	0x00000002
s0	8	0x00000002
s1	9	0x00000002
t0	5	0x00000002
t2	7	0x00000004
t2	7	0x00000008
t3	28	0x10010008
t4	29	0xffffffff
t1	6	0x00000001
t0	5	0x00000003
t2	7	0x00000006
t2	7	0x0000000c
t3	28	0x1001000c
t4	29	0x00000005

t1	6	0x00000006
t0	5	0x00000004
t2	7	0x00000008
t2	7	0x00000010
t3	28	0x10010010
t4	29	0xfffffffe
t1	6	0x00000004
t0	5	0x00000005
t2	7	0x0000000a
t1	6	0x00000007
s0	8	0x00000006
s1	9	0x00000007
t0	5	0x00000006
t2	7	0x00000018
t1	6	0x0000000b
s0	8	0x00000007
s1	9	0x0000000b
t2	7	0x0000001c
a7	17	0x0000000a

ASSIGNMENT 2 :

CODE :

```
.data
A: .word 3 5 -1 9 2          # Mảng A chứa các giá trị 3, 5, -1, 9, 2
Aend: .word                  # Địa chỉ kết thúc của mảng A
space: .asciz " "            # Chuỗi khoảng trắng
newline: .asciz "\n"         # Chuỗi xuống dòng

.text
main:
   la a0, A                  # a0 = địa chỉ của phần tử đầu tiên trong mảng
A (A[0])
   la a1, Aend               # a1 = địa chỉ kết thúc của mảng A
   addi a1, a1, -4           # a1 = địa chỉ của phần tử cuối cùng trong mảng
A (A[n-1])
   j sort                    # Nhảy đến thủ tục sắp xếp (sort)

after_sort:
   li a7, 10                 # Chuẩn bị kết thúc chương trình
   ecall                     # Gọi hệ thống để kết thúc chương trình

end_main:

# ------------------------------------------------------------
# Thủ tục sort (sắp xếp chọn tăng dần sử dụng con trỏ)
# Cách sử dụng thanh ghi trong chương trình sắp xếp:
# a0: con trỏ đến phần tử đầu tiên trong phần chưa sắp xếp
# a1: con trỏ đến phần tử cuối cùng trong phần chưa sắp xếp
# t0: biến tạm để lưu giá trị của phần tử cuối cùng
# s0: con trỏ đến phần tử lớn nhất trong phần chưa sắp xếp
# s1: giá trị của phần tử lớn nhất trong phần chưa sắp xếp
# ------------------------------------------------------------
sort:
   la a0, A                  # a0 = địa chỉ của phần tử đầu tiên trong mảng
A
   beq a0, a1, done          # Nếu danh sách chỉ có một phần tử, nó đã được
sắp xếp
   j max                     # Gọi thủ tục tìm phần tử lớn nhất (max)

after_max:
   lw t0, 0(a1)              # Load giá trị của phần tử cuối cùng vào t0
   sw t0, 0(s0)              # Gán giá trị của phần tử cuối cùng vào vị trí
của phần tử lớn nhất
   sw s1, 0(a1)              # Gán giá trị lớn nhất vào vị trí của phần tử
cuối cùng
   addi a1, a1, -4           # Giảm con trỏ đến phần tử cuối cùng
   j print_array             # In mảng sau mỗi lần sắp xếp

done:

# ------------------------------------------------------------------
# Thủ tục max
# Chức năng: tìm giá trị và địa chỉ của phần tử lớn nhất trong danh sách
# a0: con trỏ đến phần tử đầu tiên
# a1: con trỏ đến phần tử cuối cùng
```

```
# ---------------------------------------------------------------------
max:
   addi s0, a0, 0          # Khởi tạo con trỏ max trỏ đến phần tử đầu tiên
   lw s1, 0(s0)            # Khởi tạo giá trị max bằng giá trị của phần tử
đầu tiên
   addi t0, a0, 0          # Khởi tạo con trỏ next trỏ đến phần tử đầu
tiên

loop:
   beq t0, a1, ret         # Nếu next = last, trở về
   addi t0, t0, 4          # Di chuyển đến phần tử tiếp theo
   lw t1, 0(t0)            # Load giá trị của phần tử tiếp theo vào t1
   blt t1, s1, loop        # Nếu (next) < (max), lặp lại
   addi s0, t0, 0          # Phần tử tiếp theo là phần tử lớn nhất mới
   addi s1, t1, 0          # Giá trị tiếp theo là giá trị lớn nhất mới
   j loop                  # Thay đổi hoàn tất; lặp lại

ret:
   j after_max             # Trở về sau khi tìm được phần tử lớn nhất

print_array:
   la t0, A                # t0 = địa chỉ của phần tử đầu tiên trong mảng
A (A[0])
   la t1, Aend             # t1 = địa chỉ kết thúc của mảng A
   addi t1, t1, -4         # t1 = địa chỉ của phần tử cuối cùng trong mảng
A (A[n-1])

print_loop:
   lw a0, 0(t0)            # Load giá trị của phần tử hiện tại vào a0
   li a7, 1                # Chuẩn bị để in số nguyên
   ecall                   # Gọi hệ thống để in số nguyên
   li a7, 4                # Chuẩn bị để in chuỗi
   la a0, space             # In khoảng trắng
   ecall                   # Gọi hệ thống để in chuỗi
   bge t0, t1, print_done  # Nếu t0 >= t1, kết thúc in
   addi t0, t0, 4          # Di chuyển đến phần tử tiếp theo
   j print_loop            # Lặp lại vòng lặp in

print_done:
   li a7, 4                # Chuẩn bị để in chuỗi
   la a0, newline           # In xuống dòng
   ecall                   # Gọi hệ thống để in chuỗi
   j sort                  # Lặp lại quá trình sắp xếp cho danh sách nhỏ
hơn
```

Kết quả:

```
3 5 -1 2 9
3 2 -1 5 9
-1 2 3 5 9
-1 2 3 5 9
-1 2 3 5 9
```

ASSIGNMENT 3 :

```
.data
A:       .word 325, -1, 52, 1, 5, 74, -24, -2, 5, 1    # Mảng cần sắp xếp
(10 phần tử)
prompt:  .asciz " "
space:   .asciz " "            # Khoảng trắng giữa các phần tử
newline: .asciz "\n"           # Xuống dòng

.text
.globl main
main:
    # Khởi tạo: s0 = địa chỉ đầu mảng, s1 = số phần tử (10)
    la    s0, A                # s0 = địa chỉ của mảng A
    li    s1, 10               # s1 = số phần tử của mảng (10)

    # s2: chỉ số outer loop (i = 0)
    li    s2, 0                # s2 = 0 (bắt đầu từ phần tử đầu tiên)

outer_loop:
    # Nếu i >= n-1 (10-1 = 9) -> kết thúc sắp xếp
    li    t0, 9                # t0 = 9 (n-1)
    bge   s2, t0, sort_done    # Nếu s2 >= t0, nhảy đến sort_done

    # s3: chỉ số inner loop (j = 0)
    li    s3, 0                # s3 = 0 (bắt đầu từ phần tử đầu tiên)

inner_loop:
    # Tính giới hạn cho inner loop: j < (n - 1 - i)
    li    t1, 10               # t1 = 10 (số phần tử)
    sub   t1, t1, s2           # t1 = t1 - s2 (n - i)
    addi  t1, t1, -1           # t1 = t1 - 1 (n - 1 - i)
    bge   s3, t1, inner_done   # Nếu s3 >= t1, nhảy đến inner_done
```

```
    # Tính địa chỉ của A[j]: t3 = s0 + (s3*4)
    slli  t3, s3, 2           # t3 = s3 * 4 (offset của A[j])
    add   t3, s0, t3          # t3 = s0 + t3 (địa chỉ của A[j])
    lw    t4, 0(t3)           # t4 = A[j]

    # Tính địa chỉ của A[j+1]: t5 = s0 + ((s3 + 1)*4)
    addi  t5, s3, 1           # t5 = s3 + 1 (j + 1)
    slli  t5, t5, 2           # t5 = t5 * 4 (offset của A[j+1])
    add   t5, s0, t5          # t5 = s0 + t5 (địa chỉ của A[j+1])
    lw    t6, 0(t5)           # t6 = A[j+1]

    # So sánh và hoán đổi nếu cần:
    # Nếu A[j] <= A[j+1] thì không đổi, nếu A[j] > A[j+1] thì hoán đổi
    ble   t4, t6, no_swap     # Nếu t4 <= t6, nhảy đến no_swap
    sw    t6, 0(t3)           # A[j] = t6 (hoán đổi A[j] và A[j+1])
    sw    t4, 0(t5)           # A[j+1] = t4

no_swap:
    addi  s3, s3, 1           # j++
    j     inner_loop           # Lặp lại inner_loop

inner_done:
    # Sau mỗi lượt inner loop (1 pass), gọi procedure in mảng
    jal   ra, printArray      # Gọi hàm printArray để in mảng
    addi  s2, s2, 1           # i++
    j     outer_loop           # Lặp lại outer_loop

sort_done:
    # In mảng cuối cùng đã được sắp xếp
    jal   ra, printArray      # Gọi hàm printArray để in mảng
    li    a7, 10             # Chuẩn bị để kết thúc chương trình
    ecall                     # Kết thúc chương trình
```

```
#-------------------------------------------------------
# Procedure printArray:
# In ra mảng theo định dạng "Array: <A[0]> <A[1]> ... <A[9]>\n"
#-------------------------------------------------------
printArray:
    la    a0, prompt        # In chuỗi "Array: "
    li    a7, 4             # Chuẩn bị để in chuỗi
    ecall                   # Gọi hệ thống để in chuỗi

    la    t0, A             # t0 = địa chỉ đầu mảng A
    li    t1, 10            # t1 = số phần tử cần in (10)

print_loop:
    beq   t1, zero, print_done   # Nếu t1 == 0, nhảy đến print_done
    lw    t2, 0(t0)         # t2 = A[i]
    mv    a0, t2            # a0 = t2 (chuẩn bị để in số nguyên)
    li    a7, 1             # Chuẩn bị để in số nguyên
    ecall                   # Gọi hệ thống để in số nguyên

    la    a0, space         # In khoảng trắng giữa các số
    li    a7, 4             # Chuẩn bị để in chuỗi
    ecall                   # Gọi hệ thống để in chuỗi

    addi  t0, t0, 4         # Chuyển đến phần tử kế tiếp (t0 += 4)
    addi  t1, t1, -1        # Giảm số phần tử cần in (t1--)
    j     print_loop         # Lặp lại print_loop

print_done:
    la    a0, newline       # Xuống dòng sau khi in xong
    li    a7, 4             # Chuẩn bị để in chuỗi
    ecall                   # Gọi hệ thống để in chuỗi
    jr    ra                # Trở về địa chỉ được lưu trong ra
```

Kết quả:

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

-1 52 1 5 74 -24 -2 5 1 325
-1 1 5 52 -24 -2 5 1 74 325
-1 1 5 -24 -2 5 1 52 74 325
-1 1 -24 -2 5 1 5 52 74 325
-1 -24 -2 1 1 5 5 52 74 325
-24 -2 -1 1 1 5 5 52 74 325
-24 -2 -1 1 1 5 5 52 74 325
-24 -2 -1 1 1 5 5 52 74 325
-24 -2 -1 1 1 5 5 52 74 325
-24 -2 -1 1 1 5 5 52 74 325

-24 -2 -1 1 1 5 5 52 74 325
-24 -2 -1 1 1 5 5 52 74 325
-24 -2 -1 1 1 5 5 52 74 325

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

ASSIGNMENT 4:

Code:

```
.data

A:

.word 50, 12, -3, 0, 7, 98, -12, 45, 3, 27   # Bộ dữ liệu mảng mới (10 phần tử)

prompt:  .asciz " "

space:   .asciz " "

newline: .asciz "\n"

.text

.globl main

main:

   # Khởi tạo:

   # Chuỗi in đầu dòng cho mảng

   # Khoảng trắng giữa các phần tử

   # Xuống dòng

   # s0 = địa chỉ đầu mảng, s1 = số phần tử (10)

   la    s0, A

   li    s1, 10

   # s3 = chỉ số outer loop, bắt đầu từ 1 (vì phần tử A[0] được coi là đã sắp xếp)

   li    s3, 1

outer_loop:

   bge   s3, s1, sort_done    # Nếu i >= n thì mảng đã được sắp xếp

   # Lấy key = A[i]

   slli  t1, s3, 2            # t1 = i * 4 (để tính địa chỉ của A[i])

   add   t2, s0, t1           # t2 = địa chỉ của A[i]

   lw    t3, 0(t2)            # t3 = key (giá trị cần chèn)

   addi  s4, s3, -1           # s4 = j = i - 1


inner_loop:

    blt   s4, zero, inner_done   # Nếu j < 0, thoát vòng lặp bên trong

    slli  t1, s4, 2               # t1 = j * 4

    add   t2, s0, t1              # t2 = địa chỉ của A[j]
```

```
    lw    s2, 0(t2)              # s2 = A[j]
    ble   s2, t3, inner_done      # Nếu A[j] <= key, kết thúc vòng lặp bên trong
    # Dời A[j] sang phải: A[j+1] = A[j]
    addi  t1, s4, 1              # t1 = j + 1
    slli  t1, t1, 2              # t1 = (j+1) * 4
    add   t2, s0, t1             # t2 = địa chỉ của A[j+1]
    sw    s2, 0(t2)              # di chuyển giá trị A[j] sang phải
    addi  s4, s4, -1             # j = j - 1
    j     inner_loop

inner_done:
        addi  s4, s4, 1             # Vị trí chèn: j+1
   slli  t1, s4, 2             # t1 = (j+1) * 4
   add   t2, s0, t1            # t2 = địa chỉ của A[j+1]
   sw    t3, 0(t2)             # Chèn key vào A[j+1]
   # In mảng sau mỗi lượt chèn
   jal   ra, printArray
   addi  s3, s3, 1
   j     outer_loop
sort_done:
   # i = i + 1
   # In mảng cuối cùng đã được sắp xếp
   jal   ra, printArray
   li    a7, 10
   ecall
   # Kết thúc chương trình
   #-------------------------------------------------------------
   # Procedure printArray:
   # In ra mảng theo định dạng "Array: <A[0]> <A[1]> ... <A[9]>\n"
   #-------------------------------------------------------------
printArray:
   la    a0, prompt
```

```
    li    a7, 4
    ecall
    la    t0, A
    li    t1, 10
print_loop:
    # In chuỗi "Array: "
    # t0 = địa chỉ đầu mảng A
    # t1 = số phần tử cần in (10)
    beq   t1, zero, print_done  # Nếu đã in hết các phần tử
    lw    t2, 0(t0)
    mv    a0, t2
    li    a7, 1
    ecall
    # Tải phần tử hiện tại vào t2
    # In số nguyên (ECALL với a7 = 1)
    la    a0, space           # In khoảng trắng giữa các phần tử
    li    a7, 4
    ecall
    addi  t0, t0, 4
    addi  t1, t1, -1
    j     print_loop
print_done:
    # Chuyển đến phần tử kế tiếp
    la    a0, newline         # Xuống dòng sau khi in xong
    li    a7, 4
    ecall
    jr    ra
```

Kết quả chương trình:

```
12 50 -3 0 7 98 -12 45 3 27
-3 12 50 0 7 98 -12 45 3 27
-3 0 12 50 7 98 -12 45 3 27
-3 0 7 12 50 98 -12 45 3 27
-3 0 7 12 50 98 -12 45 3 27
-12 -3 0 7 12 50 98 45 3 27
```

```
-12 -3 0 7 12 50 98 45 3 27
-12 -3 0 7 12 45 50 98 3 27
-12 -3 0 3 7 12 45 50 98 27
-12 -3 0 3 7 12 27 45 50 98
-12 -3 0 3 7 12 27 45 50 98

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