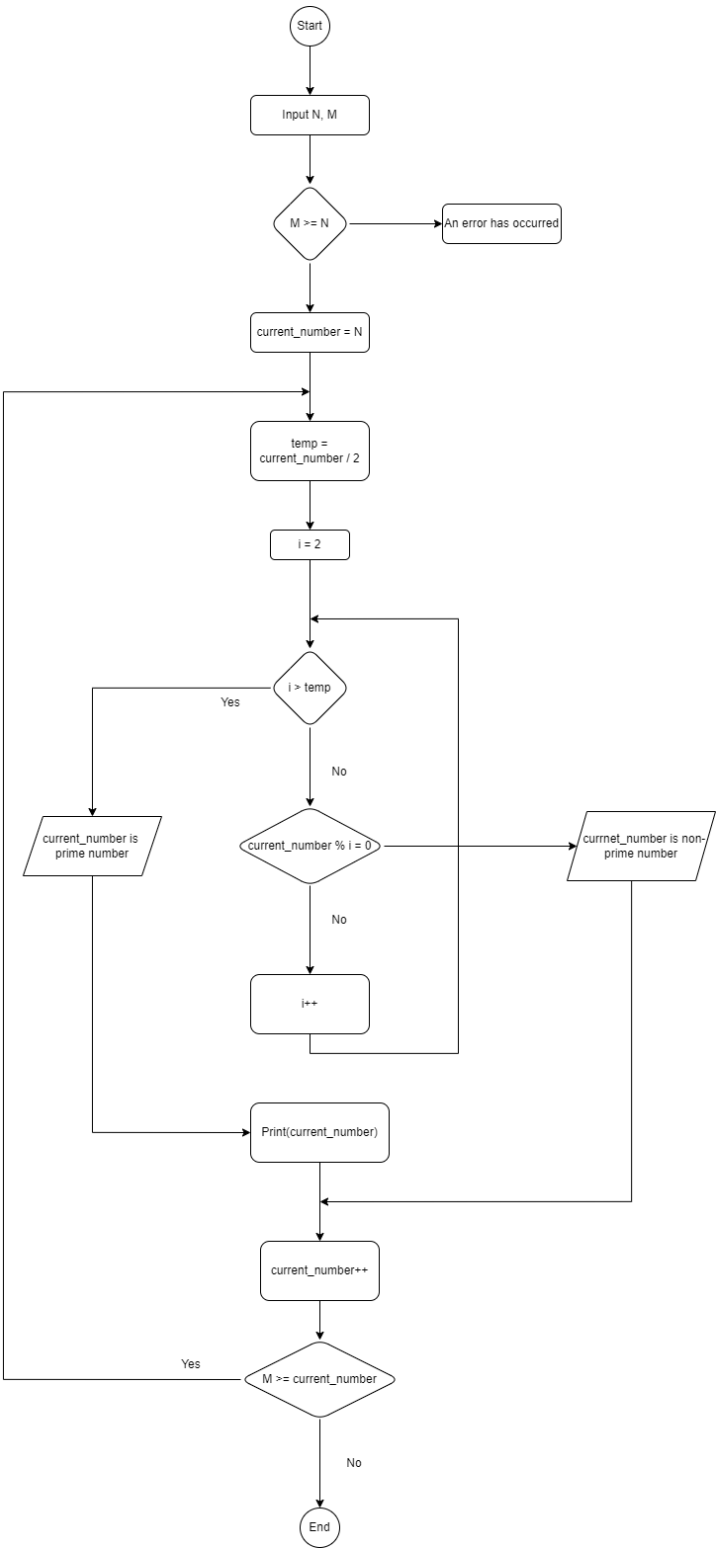
# Exercise 2:

## Đề bài:

Find all prime numbers (such as 2, 3, 5, 7, etc) in a range from the integer N to the integer M. N, M is entered from the keyboard.

## Lưu đồ thuật toán:



## Mã nguồn:

```
.data
Message1: .asciiz "Input the integer N:"
Message2: .asciiz "Input the integer M:"
Message3: .asciiz "An error has occurred (M < N)"

.text

check_inputN:
     addi $v0, $zero, 51   # Read input N
     la $a0, Message1
     syscall
     bne $a1, 0, check_inputN # if $a1 != 0: an error has occurred, branch to
        check_inputN
     nop
     add $s0, $zero, $a0   # Store N in $s0

check_inputM:
     addi $v0, $zero, 51   # Read input N
     la $a0, Message2
     syscall
     bne $a1, 0, check_inputM # if $a1 != 0: an error has occurred, branch to
        check_inputM
     nop
     add $s1, $zero, $a0   # Store M in $s1

     slt $t0, $s1, $s0  # if M < N: an error has occurred, end program
     bne $t0, 1, ok
     nop
     addi $v0, $zero, 55
     la $a0, Message3
     syscall
     j done
     nop

ok:
     add $s2, $zero, $s0   # initialize current_number = N
     slti $t0, $s2, 2   # if current_number < 2 then current_number = 2
     bne $t0, 1, main_loop
     nop
     addi $s2, $zero, 2    # current_number = 2
main_loop:
     slt $t0, $s1, $s2  # if current_number > M then end program
     beq $t0, 1, done
     nop
     jal isprime
     nop
     bne $t1, 1, continue  # continue if current_number is non-prime
     nop
     jal print_number   # if current_number is prime number, then print it
     nop
continue:
```

```
     addi $s2, $s2, 1   # current_number = current_number + 1
     j main_loop



isprime:
push:
     addi $sp, $sp, -12    # adjust the stack pointer
     sw $s0, 8($sp)     # store $s0 (N)
     sw $s1, 4($sp)     # store $s1 (M)
     sw $s2, 0($sp)     # store $s2 (current_number)

work:
     addi $t1, $zero, 1    # initialize return value = 1 (if $t1 = 1 then
        current_number is prime number)
     srl $s1, $s2, 1       # temp = current_number/2
     addi $s3, $zero, 2    # initialize i = 2
loop:
     slt $t0, $s1, $s3  # if i > temp then end procedure
     beq $t0, 1, pop
     nop

     div $s4, $s2, $s3  # $s4 = current_number / i
     mul $s4, $s4, $s3  # $s4 = $s4 * i
     slt $t0, $s4, $s2  # if ($s4 = current_number) then current_number is
        divisible by i
     bne $t0, 1, noprime   # curent_number is non-prime number
     nop
     addi $s3, $s3, 1   # i = i+1
     j loop
     nop
noprime:
     add $t1, $zero, $zero    # set $v0 = 0, end procedure
pop:
     lw $s2, 0($sp)     # restore $s2 (current_number)
     lw $s1, 4($sp)     # restore $s1 (M)
     lw $s0, 8($sp)     # restore $s0 (N)
     addi $sp, $sp, 12  # adjust the stack pointer
     jr $ra          # end procedure


print_number:
     addi $v0, $zero, 1
     add $a0, $zero, $s2
     syscall
     addi $v0, $zero, 11
     li $a0, ' '
     syscall
     jr $ra

done:
```

## Giải thích phần chương trình chính:

- Khởi tạo giá trị:
  - Chúng ta khởi tạo 3 Message như hình dưới.

```
.data
Message1: .asciiz "Input the integer N:"
Message2: .asciiz "Input the integer M:"
Message3: .asciiz "An error has occurred (M < N)"
```

- Nhập giá trị input N, M

```
check_inputN:   addi $v0, $zero, 51          # Read input N
                la $a0, Message1
                syscall
                bne $a1, 0, check_inputN     # if $a1 != 0: an error has occurred
                nop
                add $s0, $zero, $a0          # Store N in $s0
```

- Nếu giá trị của N nhập vào không phải là số tự nhiên, thì thanh ghi $a1 $\neq 0$. Lệnh bne sẽ quay lại nhãn *check_ inputN* và bắt người dùng nhập lại N đến khi thoả mãn. *(Tương tự với M)*. Giá trị của N được lưu ở thanh ghi $s0, M ở $s1.
- *Thực hiện chương trình:*
Nhập vào giá trị N = 44, M = 84. Kết quả thực hiện:

| | | |
|---|---|---|
| $s0 | 16 | 0x0000002c |
| $s1 | 17 | 0x00000054 |

- Kiểm tra điều kiện của M và N

```
                slt $t0, $s1, $s0            # if M < N: an error has occurred, end program
                bne $t0, 1, ok
                nop
                addi $v0, $zero, 55
                la $a0, Message3
                syscall
                j done
                nop
ok:
```

- Lệnh *slt* và *bne* kiểm tra giá trị của M và N, nếu người dùng nhập vào M < N thì câu lệnh *syscall* sẽ gọi đến chức năng **MessageDialog** *($v0 = 55)* để thông báo lỗi (Message3: "An error has occurred (M < N)") và nhảy đến nhãn done, kết thúc chương trình.
- Nếu người dùng nhập vào M $\geq$ N thì lệnh *bne* sẽ nhảy đến nhãn *ok*, chương trình tiếp tục được thực hiện.
- *Thực hiện chương trình:*
Giá trị của thanh ghi $t0 = 0. Chứng tỏ M $\geq$ N, chương trình nhảy đến nhãn *ok*.

| | | |
|---|---|---|
| $t0 | 8 | 0x00000000 |

- Khởi tạo giá trị *current_ number*

```
add $s2, $zero, $s0             # initialize current_number = N
slti $t0, $s2, 2                # if current_number < 2 then current_number = 2
bne $t0, 1, main_loop
nop
addi $s2, $zero, 2              # current_number = 2
```

- Khối lệnh tiếp theo thực hiện khởi tạo giá trị cho biến *current_number* (Thanh ghi $s2). Đầu tiên gán giá trị N cho *current_number*. Sau đó kiểm tra, nếu giá trị *current_number* $< 2$ thì gán *current_number* $= 2$.
- Thực hiện chương trình:
Do giá trị N $= 44 > 2$ nên *current_number* $= 2$.

| | | |
|---|---|---|
| $s2 | 18 | 0x0000002c |

- Khối lệnh chính của chương trình: *main_loop*

```
main_loop:
                slt $t0, $s1, $s2           # if current_number > M then end program
                beq $t0, 1, done
                nop
                jal isprime
                nop
                bne $t1, 1, continue        # continue if current_number is non-prime
                nop
                jal print_number            # if current_number is prime number, then print it
                nop
continue:       addi $s2, $s2, 1            # current_number = current_number + 1
                j main_loop
```

- Đầu tiên, chương trình sẽ kiểm tra nếu *current_number* $>$ M thì sẽ kết thúc chương trình.
- Nếu *current_number* $\leq$ M thì chương trình sẽ gọi đến hàm *isprime (Chi tiết về hàm isprime ở phần sau)*. Hàm *isprime* trả về giá trị nằm ở thanh ghi $t1. Nếu $t1 $= 1$ thì *current_number* là số nguyên tố, nếu $t1 $= 0$ thì *current_number* không phải là số nguyên tố.
- Nếu $t1 $= 1$ thì chương trình sẽ gọi đến thủ tục *print_number* để thực hiện in giá trị *current_number* ra màn hình.
- *Thực hiện chương trình:*

  * Giá trị hiện tại của *current_number* $= 44$. Thanh ghi $t0 có giá trị $= 0$, vậy *current_number* $\leq$ M.

    | | | |
    |---|---|---|
    | $t0 | 8 | 0x00000000 |

  * Sau khi gọi đến hàm *isprime*, thanh ghi $pc nhảy đến địa chỉ của hàm *isprime*, thanh ghi $ra lưu giá trị của địa chỉ ngay sau lệnh *jal* trong chương trình chính.

    | | | |
    |---|---|---|
    | $ra | 31 | 0x00400094 |
    | pc | | 0x004000b4 |

* Sau khi thực hiện xong hàm *isprime*, do *current_number* = 44, không phải là số nguyên tố nên thanh ghi $t1 = 0 và thanh ghi $pc nhận giá trị mà thanh ghi $ra đã cất giữ trước đó.

| | | |
|---|---|---|
| $t1 | 9 | 0x00000000 |

| | | |
|---|---|---|
| $ra | 31 | 0x00400094 |
| pc | | 0x00400094 |

- Khối lệnh *continue*

```
continue:       addi $s2, $s2, 1              # current_number = current_number + 1
                j main_loop
```

- Khối lệnh này thực hiện tăng giá trị *current_number* lên 1 đơn vị và quay lại nhãn *main_loop*.

**Kết quả thực hiện:**
Với input N = 44, M = 84

```
47 53 59 61 67 71 73 79 83
-- program is finished running (dropped off bottom) --
```

## Giải thích các thủ tục và hàm:

- Hàm *isprime:*
  Hàm *isprime* trả về giá trị nằm ở thanh ghi $t1. Nếu $t1 = 1 thì *current_number* là số nguyên tố, nếu $t1 = 0 thì *current_number* không phải là số nguyên tố.

  – Khối lệnh *push*:

```
isprime:
push:           addi $sp, $sp, -12          # adjust the stack pointer
                sw $s0, 8($sp)              # store $s0 (N)
                sw $s1, 4($sp)              # store $s1 (M)
                sw $s2, 0($sp)              # store $s2 (current_number)
```

  * Thực hiện lưu lại giá trị của 3 thanh ghi đầu vào ($s0 - N, $s1 - M, $s2 - *current_number*) vào ngăn xếp để cất giữ.

  – Khối lệnh *work*:

```
work:           addi $t1, $zero, 1          # initialize return value = 1
                srl $s1, $s2, 1             # temp = current_number/2
                addi $s3, $zero, 2          # initialize i = 2
```

  * Thực hiện khởi tạo 3 giá trị $t1 = 1 (return number), $s1 = *current_number*/2 (temp), $s3 = 2 (i).

  – Khối lệnh *loop*:

```
loop:
                slt $t0, $s1, $s3           # if i > temp then end procedure
                beq $t0, 1, pop
                nop
                div $s4, $s2, $s3           # $s4 = current_number / i
                mul $s4, $s4, $s3           # $s4 = $s4 * i
                slt $t0, $s4, $s2           # if ($s4 = current_number) then current_num
                bne $t0, 1, noprime         # curent_number is non-prime number
                nop
                addi $s3, $s3, 1            # i = i+1
                j loop
                nop
```

  * Đầu tiên, chương trình sẽ kiểm tra nếu i > temp thì sẽ nhảy đến nhãn *pop* để kết thúc hàm.
  * Thực hiện cho i chạy từ 2 -> *current_number*/2. Nếu *current_number* chia hết cho i thì chứng tỏ *current_number* không phải là số nguyên tố.
  * Tìm số dư của *current_number* / i bằng cách chia rồi lấy thương (giá trị nguyên) sau đó đem nhân lại với i. Nếu giá trị vừa nhân lên = *current_number* thì chứng tỏ *current_number* chia hết cho i, chương trình sẽ nhảy đến nhãn *noprime*. Nếu *current_number* không chia hết cho i thì tăng i lên 1 đơn vị rồi quay lại nhãn *loop*.

  – Khối lệnh *noprime* và *pop*:

```
noprime:
                add $t1, $zero, $zero       # set $t1 = 0, end procedure
pop:            lw $s2, 0($sp)              # restore $s2 (current_number)
                lw $s1, 4($sp)              # restore $s1 (M)
                lw $s0, 8($sp)              # restore $s0 (N)
                addi $sp, $sp, 12           # adjust the stack pointer
                jr $ra                      # end procedure
```

* Khối lệnh *noprime* sẽ thực hiện gán giá trị trả về $t1 = 0.
* Khối lệnh *pop* thực hiện trả lại giá trị ban đầu cho các biến đã lưu trong ngăn xếp và giải phóng ngăn xếp, gọi lệnh *jr* để quay về chương trình chính.

- Hàm *print_number*
  Hàm *print_number in ra số nguyên tố được lưu ở thanh ghi $s2*

```
print_number:
                addi $v0, $zero, 1
                add $a0, $zero, $s2
                syscall
                addi $v0, $zero, 11
                li $a0, ' '
                syscall
                jr $ra
done:
```

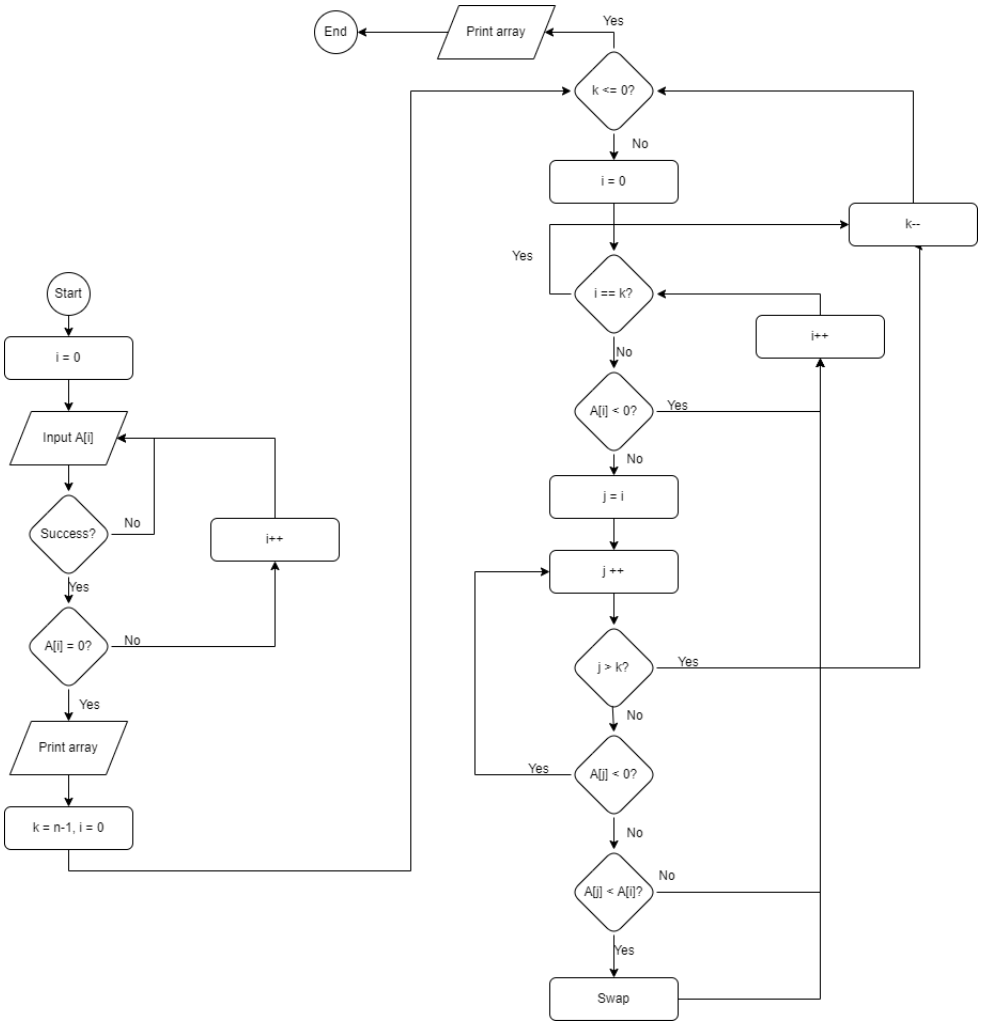
  – Đầu tiên, hàm sẽ gọi đến chức năng **print decimal integer** ($v0 = 1) để in ra số nguyên tố được lưu trong thanh ghi $s2.
  – Sau đó, hàm sẽ gọi đến chức năng **print character** ($v0 = 11) để in ra kí tự *space*, giúp ngăn cách các số nguyên tố khi liệt kê.

# Exercise 7:

## Đề bài:

Some people are standing in a row in a park. There are trees between them which cannot be moved. Your task is to rearrange the people by their heights in a non-descending order without moving the trees. People can be very tall!

## Lưu đồ thuật toán:



## Mã nguồn:

```
.data
array: .space 400  #100 elements
prompt: .asciiz "Enter an integer (0 to quit) :"
mes1: .asciiz "Before sorted:"
mes2: .asciiz "\nAfter sorted:"

.text
main:
   la $a2, array   # Load address of array

read_array:
     li $v0, 51    # Input int mode
     la $a0, prompt  # Message
     syscall

     bne $a1, 0, read_array # If $a1 != 0: an error has occurred, get input again
     nop

   beqz $a0, read_done # If A[i] = 0, exit read_array
   nop

     sw $a0, 0($a2)  # Save to A[i]
     addiu $a2, $a2, 4 # Move to next element

     j read_array    # Loop to read array
     nop

read_done:
   addi $a1,$a2,-4    # $a1 = Address(array[n-1])
   li $v0, 4     # Print string mode
   la $a0, mes1
   syscall

   la $a0,array    # $a0 = Address(array[0])
   addi $a3,$a0,0  # Mark first element
   jal print_array
   nop

   la $a0,array    # $a0 = Address(array[0])
   addi $a3,$a0,0  # Remark first element
   addi $t8,$a1,0  # Mark last element
   j sort       # sort
   nop

end:
   li $v0, 4     # Print string mode
   la $a0, mes2
   syscall

   addi $a1,$t8,0  #Load last element
   jal print_array
```

```
   li $v0, 10   #exit
   syscall

sort:
   slt $t3,$a0,$a1
   beq $t3,$zero,done #single element list is sorted
   j max      #call the max procedure
   nop

after_max:
   addi $a1,$a1,-4    # Decrement pointer to last element
   j sort       # Repeat sort for smaller list
   nop

done:
   j end
   nop

max:
   addi $v0,$a0,0  # Init pointer to first element

loop:
   slt $t3,$v0,$a1    #  If last < first
   beq $t3,$zero,ret  # If now=last, return
   lw $t1,0($v0)   # Load current element into $t1
   slt $t3,$t1,$zero # Check if (now) < 0
   bne $t3, $zero, skip # If < 0, skip this element
   nop
   addi $v1,$v0,0  # $v1 = Address(current element)

find_next:
   addi $v1,$v1,4  # Point to next element
   slt $t3,$a1,$v1    # If next element out of range
   bne $t3,$zero,ret # If out of range, return
   nop
   lw $t2,0($v1)   # Load next element into $t2
   slt $t3,$t2,$zero # Check if (next) < 0
   bne $t3, $zero, find_next # If < 0, skip this element, move to next
   nop

   slt $t0,$t2,$t1  # (next) < (now) ?
   addi $t3, $v0, 0  # Copy v0
   addi $v0, $v0, 4  # Point to next element
   beq $t0,$zero,loop  # If (next)>=(now), repeat
   sw $t2,0($t3)   # Copy A[next] to A[now]
   sw $t1,0($v1)   # Copy A[now] to A[next]
   j loop       # Change completed; now repeat
   nop

ret:
   j after_max

skip:
```

```
   addi $v0, $v0, 4  # Move to next element
   j loop        # Continue loop
   nop

print_array:
   addi $v0, $zero, 11
   li $a0, ' '
   syscall

   li $v0, 1     # Print int mode
   lw $a0,0($a3)
   syscall

   addi $a3, $a3, 4
   slt $t7, $a1, $a3
   beq $t7,$zero,print_array
   nop

   jr $ra
```

## Giải thích phần chương trình chính:

- Khởi tạo giá trị:
  - Chúng ta khởi tạo 3 Message và mảng 100 phần tử như hình dưới.

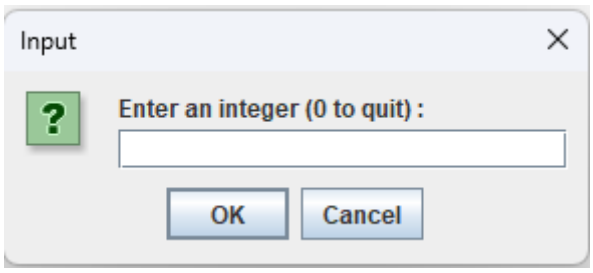
```
.data
array:   .space 400      #100 elements
prompt: .asciiz "Enter an integer (0 to quit) :"
mes1:    .asciiz "Before sorted:"
mes2:    .asciiz "\nAfter sorted:"
```

- Nhập giá trị của mảng

  Đầu tiên, load giá trị địa chỉ của mảng vào a2:


  Tiếp theo, chuyển đến khối readarray có nhiệm vụ nhận vào mảng được người dùng nhập từ bàn phím.

  Đưa thanh ghi v0 về giá trị 51 tương ứng với gọi dialog nhập số nguyên. Gán vào a0 địa chỉ của promt để in chuỗi này vào hộp thoại dialog. Sau đó gọi lệnh syscall để hiển thị hộp thoại nhập số liệu:



Khi nhập sai dạng của số liệu (VD "hello"), a1 sẽ nhận giá trị -1:

| | | |
|---|---|---|
| $a1 | 5 | 0xffffffff |

Tương tự, giá trị này sẽ được gán như sau:

- 0: Giá trị nhập vào đúng dạng
- -1: Có giá trị được nhập nhưng sai kiểu dữ liệu
- -2: Người dùng chọn Cancel
- -3: Người dùng chọn OK nhưng giá trị để trống

Vì vậy, sau khi nhận giá trị, ta kiểm tra giá trị nhập vào đã thoả mãn hay chưa nếu chưa thì cho quay lại khối read_array để nhập lại.

Nếu kiểu dữ liệu đã thoả mãn và thấy giá trị bằng 0, ta kết thúc nhập dữ liệu, rẽ nhánh sang khối read_done. Còn ngược lại, lưu giá trị này vào mảng, sau đó tăng giá trị biến chạy thêm 4 để trỏ đến phần tử tiếp theo và tiếp tục đọc từ bàn phím.

Sau khi đọc xong mảng, ta thu được kết quả:


- Sau khi đọc xong mảng

  Tiếp theo, ta chuyển đến khối read_done:

  Đầu tiên, ta đưa giá trị $v0 về 4 và gán $a0 địa chỉ của chuỗi mes1 để in chuỗi này ra màn hình:

  ```
  Before sorted:
  ```

  Sau đó, ta gán $a3 là địa chỉ của mảng, cũng là địa chỉ của phần tử đầu tiên, $a1 lưu giá trị địa chỉ cuối cùng của mảng để đánh dấu vị trí cần in ra.

| | | |
|---|---|---|
| $a1 | 5 | 0x10010018 |
| $a2 | 6 | 0x1001001c |
| $a3 | 7 | 0x10010000 |

  Sau khi đã đánh dấu, ta gọi hàm print_array, dung lệnh jal để đánh dấu vị trí tiếp tục chương trình sau khi kết thúc hàm:

| | | |
|---|---|---|
| $ra | 31 | 0x00400060 |
| pc | | 0x0040012c |

- In mảng ra màn hình

  Đầu tiên, ta chuyển $v0 về giá trị 11 để in ra ký tự space ngăn cách giữa các phần tử của mảng. Sau đó, ta lại chuyển giá trị này về 1, load giá trị của $a3 vào $a0 để in nó ra màn hình.

  Tiếp theo, ta tăng giá trị của $a3 thêm 4 để trỏ đến phần tử tiếp theo, nếu phần tử này nằm ngoài phạm vi của mảng, có giá trị lớn hơn địa chỉ của phần tử cuối cùng trong mảng, ta biết mảng đã được in hết nên sẽ nhảy đến vị trí tiếp theo của chương trình đang được lưu trong thanh ghi $ra.

Kết quả sau khi in mảng ra màn hình:

```
Before sorted: -1 150 190 170 -1 160 180
```

- Kết thúc in mảng

  Khi trở lại khối lệnh chính, ta lấy lại các giá trị $a0 lưu địa chỉ mảng và copy giá trị này vào $a3, tương tự, ta copy thêm $a1 vào $t8 để lưu địa chỉ phần tử cuối cùng của mảng.

  Sau khi đã lưu lại các giá trị cần thiết, ta thực hiện việc sắp xếp lại mảng:

- Sắp xếp mảng

  $a0 lưu địa chỉ của A[0] ,$a1 lưu địa chỉ của A[n-1].

  Sau đó kiểm tra 2 giá trị này, nếu $a0 >= $a1 => mảng chỉ có 1 phần tử hoặc đã xét đến phần tử cuối cùng => Kết thúc chương trình.

  Ở đây, $a0 < $a1 => chương trình tiếp tục, nhảy đến khối max:

  Tiếp theo, lưu giá trị $a0 vào $v0 để làm biến chạy:

| | | |
|---|---|---|
| $v0 | 2 | 0x10010000 |
| $v1 | 3 | 0x00000000 |
| $a0 | 4 | 0x10010000 |

Khi bắt đầu vòng lặp, ta kiểm tra biến chạy đã chạy đến phần tử cuối cùng của mảng chưa, nếu nó có giá trị trùng với $a1, tức là ta đã xét đến phần tử cuối cùng và nhảy đến khối ret. Ở đây, ta bắt đầu xét từ phần tử đầu tiên nên điều kiện này sai, ta sẽ tiếp tục thực hiện chương trình.

Ta thực hiện load giá trị của phần tử đang xét vào $t1:

| | | |
|---|---|---|
| $t1 | 9 | 0xffffffff |

Sau đó kiểm tra giá trị này, nếu <0, tương ứng với vị trí của một cái cây, ta nhảy đến khối skip để tăng giá trị biến chạy thêm 4 và quay lại vòng lặp như trên.

Khi xét đến A[1] = 0x96 (150) > 0, ta tiếp tục chương trình, copy giá trị của $v0 vào $v1.

Khối find_next thực hiện tìm phần tử >0 ngay sau phần tử đang xét. Khi bắt đầu vòng lặp, thực hiện tăng $v1 lên 4 để trỏ đến phần tử tiếp theo, so sánh với $a1 để đảm bảo không vượt phạm vi của mảng.

Nếu $v1 > $a1, tức phần tử $v0 đã là phần tử cuối cùng của mảng, ta chuyển đến khối ret

Ở đây, ta tìm được giá trị tiếp theo là 190, lưu giá trị này vào $t2:

| | | |
|---|---|---|
| $t1 | 9 | 0x00000096 |
| $t2 | 10 | 0x000000be |

Nếu phần tử này <0, tương ứng với vị trí của cây, ta quay lại khối find_next để xét các phần tử tiếp theo trong mảng.

Sau khi tìm được 2 phần tử liền kề, dương trong mảng, ta tiến hành so sánh 2 giá trị. Nếu A[j+1] >= A[j] => tăng giá trị $v0 thêm 4 để đưa biến chạy đến phần tử tiếp theo

và quay lại vòng lặp mà không làm gì. Ngược lại, thực hiện swap 2 giá trị. Ở đây, 190 > 150, ta chỉ quay lại vòng lặp.

Chương trình tiếp tục đến khi $v0 lưu địa chỉ A[2] = 190, $v1 lưu địa chỉ A[3] = 170. Hai lệnh sw lưu chéo giá trị của 2 biến này cho nhau:

| Value (+8) | Value (+c) |
|---|---|
| 0x000000aa | 0x000000be |

Sau đó tiếp tục quay lại vòng lặp.

Kết thúc lần lặp đầu tiên, khi $v0 = $a1:

| | | |
|---|---|---|
| $v0 | 2 | 0x10010018 |
| $v1 | 3 | 0x10010018 |
| $a0 | 4 | 0x10010000 |
| $a1 | 5 | 0x10010018 |

Ta đã xét đến phần tử cuối cùng. Khi này, phần tử lớn nhất 0xbe đã được đẩy đến cuối cùng:

| Address | Value (+0) | Value (+4) | Value (+8) | Value (+c) | Value (+10) | Value (+14) | Value (+18) |
|---|---|---|---|---|---|---|---|
| 0x10010000 | 0xffffffff | 0x00000096 | 0x000000aa | 0x000000a0 | 0xffffffff | 0x000000b4 | 0x000000be |
| 0x10010020 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |

Ta trừ $a1 đi 4 để trỏ đến phần tử A[n-2] và tiếp tục quay lại khối loop để sắp xếp tiếp mảng còn lại.

Vòng lặp kết thúc khi $a1 = $a0, tất cả phần tử đều đã được sắp xếp.

Khi đã sắp xếp xong, ta tiếp tục thực hiện syscall tương tự ở trên để in chuỗi ra màn hình:

```
After sorted:
```

Sau đó ta load lại địa chỉ phần tử cuối mảng đã được copy vào $t8 từ trước để trả lại cho $a1, cùng với $a3 vẫn lưu địa chỉ của mảng để gọi hàm print_array, in lại mảng sau khi đã được sắp xếp:

```
After sorted: -1 150 160 170 -1 180 190
```

Cuối cùng, gán $v0 = 10 và gọi syscall để kết thúc chương trình.