MINI PROJECT – REPORT

Thực hành kiến trúc máy tính - IT4182
Đề tài (13) , (14)

Lê Văn Duẩn

Nhóm 12 – Lớp 130999

Lê Văn Duẩn - 20194508

Nguyễn Hải Dương - 20194530

1. **Đề tài (13)**

   1.1. Nội dung yêu cầu

   Ticket numbers usually consist of an even number of digits. A ticket number is considered lucky if the sum of the first half of the digits is equal to the sum of the second half. Given a ticket number n, determine if it's lucky or not.

   Example

   For n = 1230, the output should be isLucky(n) = true;

   For n = 239017, the output should be isLucky(n) = false.

   1.2. Thuật toán

   Mô tả bằng C:

```
int main() {
   stack_type stack;
   int number;
   printf("Enter ticket number: \n");
   scanf("%d",&number);
```

```
   check_input(number, stack);
   is_lucky(stack);
   return 0;
}
```

Các hàm con:

```
// Kiểm tra các điều kiện đầu vào và tách các chữ số của number và push vào stack
void check_input(int number, stack_type stack) {
   int so_du;

   if(number > MAX_LENGTH)
      printf("Number is too big");
   else if(number < 0)
      printf("Number is negative!");
   while (number > 0)
   {
      so_du = number % 10;
      stack_push(stack, so_du);
      number = number/10;
   }

   if(stack_size(stack)/2 != 0)
      printf("Number of digits is odd!");
}

// Duyệt lần lượt 2 nửa stack và tình tổng 2 nửa, so sánh và đưa ra kết quả
void is_lucky(stack_type stack) {
   int sum1 = 0, sum2 = 0;
   for (int i = 0; i < stack_size(stack)/2; i++)
```

```
    sum1+= stack_pop(stack);
  for (int j = 0; j < stack_size(stack)/2; j++)
    sum2+= stack_pop(stack);

  if(sum1 == sum2)
    printf("Lucky number");
  else printf("Not Lucky number");
}
```

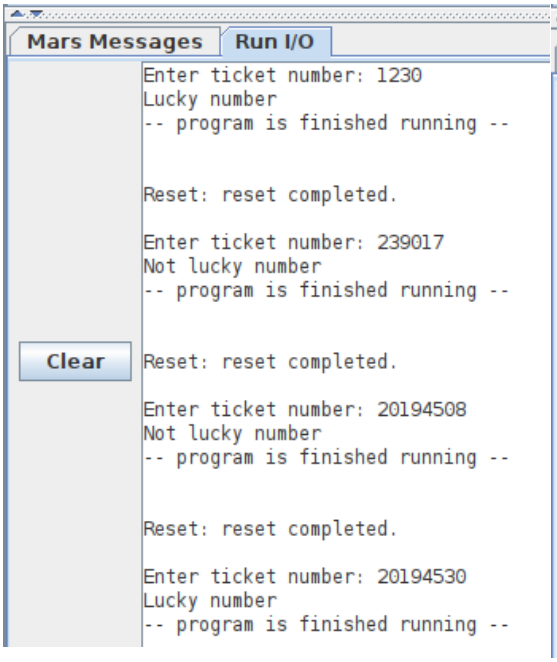
1.3. Mã nguồn

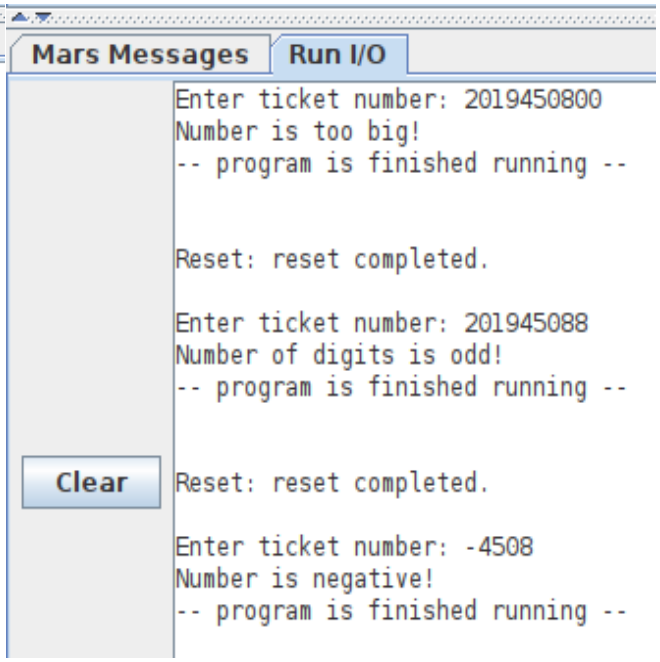
```
.data
message_input: .asciiz "Enter ticket number: "

invalid_big:      .asciiz       "Number is too big!"
invalid_negative: .asciiz       "Number is negative!"
invalid_odd:      .asciiz       "Number of digits is odd!"

lucky: .asciiz  "Lucky number"
not_lucky: .asciiz "Not lucky number"

.text
main:   li $v0, 4               # in thông báo nhận đầu vào
        la $a0, message_input
        syscall

        li $v0, 5               # Đọc số n nhập từ bàn phím
        syscall
        add $s0, $v0, $0        # Lưu giá trị n -> s0

        jal check_input         # check_input -> kiểm tra đầu vào n
        nop

        jal is_lucky            # is_lucky -> kiểm tra xem n lucky hay không và in kết quả
        nop

end_main:       li $v0, 10      # kết thúc chương trình
                syscall
#---------------------------------------------------------------------------
# function error($a1)
# In thông báo lỗi
# $a1: lưu địa chỉ của string chứa nội dung lỗi
#---------------------------------------------------------------------------
error:  li $v0, 4
        add $a0, $a1, $0
        syscall
        j end_main

```

```
#-------------------------------------------------------------------------------
# function result($a1)
# In thông báo kết quả, là hàm con trong is_lucky
# $a1: lưu địa chỉ của string chứa nội dung kết quả
#-------------------------------------------------------------------------------
result: li $v0, 4
        add $a0, $a1, $0
        syscall

        j end_is_lucky
#-------------------------------------------------------------------------------
# function check_input($s0)
# Kiểm tra số n nhập vào thỏa mãn điều kiện đầu vào hay không
# Các lỗi kiểm tra: số quá lớn, số âm, số chữ số của số là lẻ
# Đồng thời push các chữ số vào stack với con trỏ $sp
# $s0: chứa giá trị n
# Trả về: $k0 -> số chữ số của n
#         $k1 -> Một nửa số chữ số của n (1/2 của $k0)
#-------------------------------------------------------------------------------
check_input:    add  $s1, $s0, $0       # $s1 = $s0 = n
                li   $t2, 1000000000    # 1.000.000.000 -> giới hạn của n

# check_big -> kiểm tra n ($s1) < 1.000.000.000 hay không
check_big:      slt  $t1, $s1, $t2      # Nếu n($s1) < 1.000.000.000 ? t1 = 1 ngược lại t1 = 0

                la   $a1, invalid_big   # gán địa chỉ string chứa lỗi vào $a1 để sử dụng hàm error
                beqz $t1, error         # in lỗi nếu t1 = 0 ( hay n > 1.000.000.000 )

# check_negative -> kiểm tra n là số âm hay không
check_negative: slt  $t1, $0, $s1               # kiểm tra 0 < n($s1) ? t1 = 1 : t1 = 0

                la   $a1, invalid_negative      # gán địa chỉ string chứa lỗi vào $a1 để sử dụng hàm error
                beqz $t1, error                 # in lỗi nếu t1 = 0 ( hay n < 0 )

# check_odd -> kiểm tra số chữ số của n chẵn hay lẻ
check_odd:      addi $t1, $0, 10        # $t1 = 10 -> dùng làm số chia để tách các chữ số

# loop -> chia n dần cho 10 và gán lại n bằng thương và số dư push dần vào stack
# Vòng lặp dừng khi n = 0 -> hết chữ số
# $k0 đếm số chữ số
loop:           beq $s1, 0, countinue_check     # if ($s1) == 0 -> dừng loop
                nop

                divu $s1, $t1           # chia n cho 10 lấy thương và số dư ở lo, hi
                mfhi $t2                # số dư ở hi được lưu vào $t2 = n % 10
                mflo $t3                # thương ở lo được lưu vào $t3 = n / 10
                add  $s1, $t3, $0       # n($s1) = n / 10 = $t3

# push: push chữ số hay số dư vừa tìm được vào stack
push:           addi $sp, $sp, -4       # dành stack cho một phần tử
                sw   $t2, 0($sp)        # lưu số dư hay các chữ cái của n vào stack
                add  $k0, $k0, 1        # tăng biến đếm $k0 ++
                j loop

# countinue_check -> sau khi dừng vòng lặp, tiếp tục kiểm tra số chữ số lẻ hay không (kiểm tra $k0 chia hết cho 2)
countinue_check:addi $t4, $0, 2         # gán t4 = 2 -> làm số chia

                div  $k0, $t4           # chia $k0 cho 2 -> thương và dư lưu ở lo, hi
                mfhi $t4                # lấy số dư từ hi lưu vào $t4
                mflo $k1                # thương là 1/2 số chữ số của n lưu vào $k1

                la   $a1, invalid_odd   # gán địa chỉ string chứa lỗi vào $a1 để sử dụng hàm error
                bne  $t4, 0, error      # nếu không chia hết hay số dư ($t4) khác 0 -> báo lỗi bằng hàm error
                nop

                jr $ra                  # thoát hàm check_input trở về hàm main
```

```
#-------------------------------------------------------------------------
# function is_lucky($sp, $k1)
# Kiểm tra tổng nữa đầu và nửa sau của các chữ số của n (lucky) và in kết quả qua hàm result
# Duyệt nửa stack đầu và tính tổng, tương tự với nửa sau
# So sánh 2 tổng và đưa ra kết luận
# $sp -> stack chứ các chữ số của n
# $k1 -> 1/2 số chữ số của n hay 1/2 số phần tử của stack
#-------------------------------------------------------------------------
is_lucky: addi $t1, $0, 0       # i = 0 -> biến chạy vòn lặp nửa đầu
          addi $t2, $0, 0       # j = 0 -> biến chạy vòn lặp nửa sau
          add $s2, $0, $0       # $s2 lưu sum của nửa đầu
          add $s3, $0, $0       # $s3 lưu sum của nửa sau

# loop1: tính tổng nửa đầu của stack
loop1:  beq     $t1, $k1, loop2         # nếu i($t1) == $k0 -> dừng vòng lặp -> đi đến tính tổng nửa sau
        nop
# pop1: lấy phần tử trong stack ra để cộng dần vào $s2 để tính tổng
pop1:   lw      $t3, 0($sp)     # pop chữ ra khỏi stack vào $t3
        addi    $sp, $sp, 4     # xoá 1 mục ra khỏi stack

        add     $s2, $s2, $t3   # tính tổng ($s2) += $t3 (chữ số vừa pop ra)
        addi    $t1, $t1, 1     # tăng biến đếm để lặp i++
        j       loop1

# loop2: tính tổng nửa sau của stack
loop2:  beq     $t2, $k1, check_lucky   # nếu i($t1) == $k0 -> dừng vòng lặp -> đi đến so sánh 2 tổng ở check_lucky
        nop
# pop1: lấy phần tử trong stack ra để cộng dần vào $s3 để tính tổng
pop2:   lw      $t3, 0($sp)     # pop và lưu vào $t3
        addi    $sp, $sp, 4     # xóa 1 mục ra khỏi stack

        add     $s3, $s3, $t3   # tính tổng ($s3) += $t3
        addi    $t2, $t2, 1     # tăng biến đếm để lặp j++
        j       loop2
# check_lucky -> kiểm tra tổng 2 nửa có bằng nhau hay không
check_lucky:    la $a1, not_lucky       # lưu địa chỉ thông báo vào $a1 để sử dụng hàm result
                bne $s2, $s3, result    # nếu 2 nửa không bằng nhau in thông báo not_lucky
                nop                     # ngược lại -> lucky
                la $a1, lucky           # lưu địa chỉ thông báo vào $a1 để sử dụng hàm result
                j  result
end_is_lucky:   jr $ra                  # kết thúc hàm is_lucky trở về hàm main

```

1.4. Kết quả, giải thích





Ý nghĩa các thanh ghi:

Stt	Thanh ghi	Ý nghĩa
1	$v0	Lưu code để gọi hàm syscall như 4 -> in string, 5 -> đọc vào integer, ...
2	$a0	Lưu địa chỉ của string thông báo khai báo ở .data
3	$a1	Trong hàm result() -> chứa địa chỉ string thông báo kết quả
4	$s0	Lưu giá trị của n hay chính là số của vé số
5	$t1, $t2, $t3, $t4	Các thanh ghi làm biến trung gian, biến chạy
6	$sp	Lưu địa chỉ stack sử dụng trong hàm
7	$k0	Lưu số chữ số của n
8	$k1	Lưu số chữ số của n / 2 ($k0 / 2)
9	$s2, $s3	Lưu tổng nửa đầu, nửa sau của số các chữ số n
10	$ra	Lưu địa chỉ lệnh tiếp theo khi đi vào các thủ tục

2. Đề tài (14)

2.1. Nội dung yêu cầu:

Given two strings, find the number of common characters between them.

Example: For s1 = "aabcc" and s2 = "adcaa", the output should be commonCharacterCount(s1, s2) = 3. Strings have 3 common characters - 2 "a"s and 1 "c".

2.2. Thuật toán

Mô tả bằng C:

```
int main()
{
   stack_type stack;
   char s1[100], s2[100];

   get_input(s1);
   get_input(s2);

   common_char(s1, s2, stack);

   count_common(s1, s2, stack);
   return 0;
}
```

Các hàm con:

```
// Nhập chuỗi từ bàn phím
void get_input(char* s) {
   printf("Enter string: ");
   gets(s);
}

// Tìm các kí tự chung của 2 chuỗi và lưu vào stack
// Duyệt từng kí tự s1 nếu trùng với 1 kí tự nào trong s2 -> kiểm tra
// xem đã có kí tự này trong stack hay chưa -> chưa -> push
//                                          -> ngược lại-> kí tự s1 kế tiếp
void common_char(char* s1, char* s2, stack_type stack) {
   int i = 0;
```

```
    while(s1[i] != '\0')
    {
        int j = 0;

        while (s2[j] != '\0')
        {
            if(s1[i] == s2[j])
            {
                if(is_has_in_stack(stack, s1[i]) == false)
                    push(stack, s1[i]);
                else break;
            }
            j++;
        }
        i++;
    }
}

// Đếm tổng số lần kí tự chung xuất hiện trong cả 2 chuỗi của các kí
// tự chung và in kết quả -> result
// duyệt stack -> đếm số lần phần tử pop ra lặp trong s1, s2
// chọn count nhỏ hơn -> tăng biến đếm count tổng (result)
void count_common(char* s1, char* s2, stack_type stack) {
    int result = 0;
    for(int i = 0; i < stack_size(stack); i++){
        int count1 = count_in_string(stack_peek(stack), s1);
        int count2 = count_in_string(stack_peek(stack), s2);

        if(count1 > count2)
            result += count2;
        else result += count1;
    }
    printf("Common characters of Strings: %d", result);
}
```

```
// Nhận vào 1 chuỗi và 1 tham số (là phần từ pop ra từ stack)
// duyệt s và đếm số lần element lặp lại -> count
// trả về count
int count_in_string(int element, char* s) {
   int i = 0, count = 0;
   while(s[i] != '\0') {
      if(s[i] == element) {
         count++;
      }
      i++;
   }
   return count;
}
```

2.3. Mã nguồn

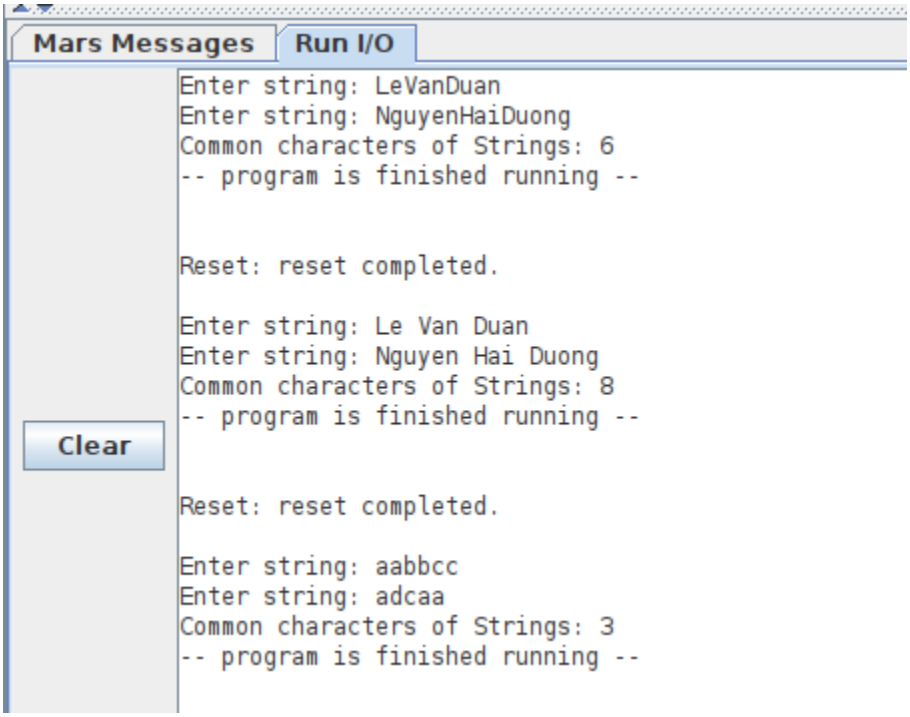
```
.data
    s1: .space 100
    s2: .space 100
    message:      .asciiz       "Enter string: "
    mess_result:  .asciiz       "Common characters of Strings: "
.text
main:   la      $t1, s1         # lưu địa chỉ chuỗi s1 để lưu nội dung chuỗi sẽ được nhập vào
        jal     get_input       # get_input($t1)-> yêu cầu nhập chuỗi từ bàn phím và lưu vào $t1
        nop
        add     $s1, $t1, $0    # địa chỉ chuỗi s1 -> $s1 = $t1

        la      $t1, s2         # lưu địa chỉ chuỗi s1 để lưu nội dung chuỗi sẽ được nhập vào
        jal     get_input       # get_input($t1)-> yêu cầu nhập chuỗi từ bàn phím và lưu vào $t1
        nop
        add     $s2, $t1, $0    # địa chỉ chuỗi s2 -> $s2 = $t1

        add     $a0, $s1, $0    # gán địa chỉ chuỗi s1($s1) vào $a0 để dùng hàm common_char
        add     $a1, $s2,$0     # gán địa chỉ chuỗi s2($s2) vào $a1 để dùng hàm common_char
        jal     common_char     # common_char($a0, $a1) -> tìm các kí tự chung và lưu vào stack
        nop

        jal     count_common    # count_common($a0, $a1, $k1) đếm số lần lặp lại của các kí tự chung
        nop
end_main:       li $v0, 10      # kết thúc chương trình
                syscall

#---------------------------------------------------------------------------
#---------------------------------------------------------------------------
# function get_input($t1)
# In thông báo và yêu cầu nhập chuỗi
# $t1: lưu địa chỉ của string để lưu chuỗi sẽ được nhập vào
#---------------------------------------------------------------------------
get_input: la $a0, message      # Load và in ra thông báo yêu cầu nhập chuỗi
           li $v0, 4            # print string
           syscall

           li $v0, 8            # read string
           add $a0, $t1, $0     # lưu space của string ($t1) vào $a0
           li $a1, 100          # độ dài lớn nhất của chuỗi nhập vào

           add $t1, $a0, $0     # lưu dịa chỉ string vừa nhập vào $t1
           syscall
           jr $ra               # kết thúc hàm và trở lại hàm main

```

```
#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
# function common_char($a0, $a1)
# Tìm tất cả các kí tự chung của 2 chuỗi có địa chỉ lưu trong $a0, $a1
# $a1, $a0: địa chỉ của chuỗi đưa vào
# Trả về: $sp -> danh sách các kí tự chung
#         $k1 -> số lượng kí tự chung
#-------------------------------------------------------------------------------
common_char:    li $t0, 0       # $t0 = i -> biến chạy

# loop1: Lặp qua các kí tự của string1 ($a0)
loop1:          add $t2, $a0, $t0       # t2 = a0 + i = address string1[i]
                lb $t3, 0($t2)          # t3 = string1[i]
                beq $t3, 10, end_common_char #  nếu là kí tự '\0' = 10 thi dừng vì hết chuỗi
                nop
                li $t1, 0       # $t1 -> j biến chạy vòng lặp 2

# loop2: với mỗi kí tự đọc từ string1($a0) , so sánh lần lượt với các kí tự string2 ($a1)
#       nếu kí tự là chung thì push vào stack, nếu không trùng tiếp tục đến kí tự tiếp theo ở string1
#       là vòng lặp lồng trong loop1
loop2:          add $t4, $a1, $t1               # t4 = a1 + j = address string2[j]
                lb $t5, 0($t4)                  # t5 = string2[j]
                beq $t5, 10, countinue_loop1    # nếu là kí tự '\0' = 10 thi dừng vì hết chuỗi
                nop
                beq $t3, $t5, push_to_stack     # nếu string1[i] == string2[j] -> push vào stack
                nop

                addi $t1,$t1,1                  # j++
                j loop2

countinue_loop1:        addi $t0, $t0, 1        # i++
                        j loop1

end_common_char:        jr $ra
```

```
#-------------------------------------------------------------------------
# push_to_stack($t3)
# Kiểm tra giá trị trong thanh ghi $t3 đã tồn tại trong stack hay chưa.
#       Nếu chưa tồn tại -> push vào stack
#       Nếu đã tồn tại -> dừng và trở lại lopp1 để đọc tiếp kí tự
# $t3: lưu giá trị của kí tự cần push vào stack
# $k1: số lượng phần tử hiện tại của stack
#-------------------------------------------------------------------------
push_to_stack:  li $t6, 0               # biến chạy $t6: k = 0

# check_loop: duyệt stack và kiểm tra xem $t3 đã có trong stack hay chưa?
check_loop:     beq $t6, $k1, push      # nếu $t6 = $k1 -> duyệt hết stack -> chưa có trong stack -> push
                nop

                sll $t7, $t6, 2                 # $t7 = $t6 * 4
                add $t7, $t7, $sp               # $t7 = $sp + k*4 = address phần tử thứ k của stack
                lb $t8, 0($t7)                  # $t8 lưu phần tử thứ k của stack
                beq $t8, $t3, countinue_loop1   # neu $t8 = $t3 ->  $t3 đã tồn tại -> không push -> thoát và quay lại vòng lặp ngoài
                nop

                addi $t6, $t6, 1                # k++
                j check_loop

# push: lưu giá trị của kí tự $t3 vào stack $sp
push:           addi $sp, $sp, -4       # để dành một phần tử trong stack
                sw   $t3, 0($sp)        # lưu $t3 vào vị tí nói trên
                addi $k1, $k1, 1        # cập nhật phần tử của stack $k1++
                j countinue_loop1
```

```
#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
# function count_common($a0, $a1, $sp)
# Đếm các số lần lặp của kí tự chung của 2 chuỗi và in ra kết quả
#       $a1, $a0: địa chỉ của chuỗi đưa vào
#       $sp -> stack các kí tự chung
#       $k1 -> số lượng kí tự chung hay số lượng hiện tại của stack
#-------------------------------------------------------------------------------
count_common:   li $t1, 0        # biến lặp i = 0
                add $t9, $ra, $0 # lưu địa chỉ trả về $t9 = $ra

# loop_stack: duyệt các phần tử của stack
loop_stack:     beq $t1, $k1, print_result      # nếu i == số lượng phần tử stack -> dừng và in kết quả
                nop

                sll $t2, $t1, 2         # t2 = t1 * 4 = 4i
                add $t2, $t2, $sp       # t2 = sp + i*4 = address phần tử thứ i của stack
                lb $s3, 0($t2)          # load phần tử thứ i vào $s3


                add $a2, $s1, $0        # gán địa chỉ string s1(đang lưu trong $s1) vao $a2 để dùng hàm count_in_string
                jal count_in_string     # count_in_string($s3, $a2) đếm số kí tự $s3 trong string $a2
                nop                     # return kết quả ra $k0

                add $s5, $k0, $0        # gán count vừa tìm được trong string s1 vao $s5

                add $a2, $s2, $0        # gán địa chỉ string s2(đang lưu trong $s2) vao $a2 để dùng hàm count_in_string
                jal count_in_string

                add $s6, $k0, $0        # gán count vừa tìm được trong string s2 vao $s6

# result: so sánh $s5, $s6 lấy giá trị nhỏ hơn và cộng dần vào kết quả $s7
result:         slt  $t3, $s5, $s6      # nếu $s5 < $s6 -> đúng $t3 = 1, ngược lại $t3 = 0
                beqz $t3, update        # nếu $t3 = 0 -> $s6 < $s5 cập nhật $s7 bằng $s6

                add $s7, $s7, $s5       # TH $t3 = 1 -> $s5 < $s6 -> cập nhật $s7

countinue_loop: addi $t1, $t1, 1        # i++
                j loop_stack            # tiếp tục lòng lặp các kí tự đang chứa trong stack

# printf_result: in ra màn hình kết quả được lưu trong $s7
print_result:   li, $v0, 4
                la $a0, mess_result
                syscall

                li $v0, 1
                add $a0, $s7, $0
                syscall

                add $ra, $t9, $0        # Khôi phục địa chỉ trả về $ra = $t9
                jr $ra

# update: cập nhật $s7 bằng $s6
update:         add $s7, $s7, $s6

                j countinue_loop

#-------------------------------------------------------------------------------
# function count_in_string($s3, $a2) đếm số kí tự $s3 trong string $a2
#       $a2: địa chỉ của chuỗi đưa vào
#       $s3 -> chứa kí tự cần đếm trong $a2
#       $k1 -> số lượng kí tự chung hay số lượng hiện tại của stack
# Trả về: $k0 -> chứa count tìm được
#-------------------------------------------------------------------------------
count_in_string:        li $t3, 0       # biến chạy i = 0
                        li $k0, 0       # gán lại giá trị trả về $k0 = 0

# loop_string: duyệt string và tăng đếm $k0 nếu có kí tự trùng với $s3
loop_string:            add $t4, $a2, $t3       # $t4 = $a2 + i = address string[i]
                        lb $t5, 0($t4)          # $t5 chứa giá trị string[i]

                        beq $t5, 10, end_count_in_string        # nếu kí tự là '\0' -> dừng lặp string -> thoát hàm

                        bne $s3, $t5, countinue_count           # Nếu kí tự dang duyệt = kí tự $s3 -> tăng count $k0
                                                                # -> ngược lại tiếp tục duyệt đến kí tự tiếp theo

                        addi $k0, $k0,1                         # count++

countinue_count:        addi $t3, $t3,1         # i++
                        j loop_string

end_count_in_string:    jr $ra  # Thoát hàm
```

2.4. Kết quả, giải thích



Ý nghĩa các thanh ghi:

Stt	Thanh ghi	Ý nghĩa
1	$v0	Lưu code để gọi hàm syscall như 4 -> in string, 5 -> đọc vào integer, ...
2	$a0	Lưu địa chỉ của string thông báo khai báo ở .data
3	$s1, $s2	Lưu địa chỉ của 2 string nhập vào
4	$a0, $a1	Sử dụng làm trung gian lưu địa chỉ 2 String s1, s2 và làm đầu vào của hàm common_char, count_common
5	$t0, $t1, $t2, $t3, $t4, $t5, $t6, $t7, $t8, $t9	Các thanh ghi làm biến trung gian, biến chạy $t9 còn được sử dụng để lưu địa chỉ $ra khi có thủ tục lồng nhau, cụ thể hàm count_common khi có hàm con count_in_string

6	$sp	Lưu địa chỉ stack sử dụng trong hàm lưu các kí tự chung giữa 2 string
7	$k0	Lưu giá trị trả về của hàm count_in_string
8	$k1	Lưu số phần tử của stack
9	$s5, $s6	Lưu số lần lặp lại của kí tự trong string s1, s2 thực hiện trong hàm count_common
10	$s7	Lưu kết quả của chương trình
11	$ra	Lưu địa chỉ lệnh tiếp theo khi đi vào các thủ tục