Aras Valizadeh 401243095

1)

در ابتدا بخش دیتا سگمنت برنامه پیاده سازی شده بدین صورت که برای گرفتن رشته ورودی یک فضا ۲۵۶ بایتی در حافظه با اسم space نگه داری می‌کنیم و در ادامه ۳ پیامی که در طول برنامه برای کاربر چاپ می‌شود به صورت asciiz نگه داری می‌شود .

```
        .data
buffer: .space 256
prompt : .asciiz  "enter your parentheses\n\0"
Yes : .asciiz  "Yes\0"
No : .asciiz "No\0"
```

در ادامه به پیاده سازی بخش text پرداخته ایم که با تعریف main شروع می شود. که در ابتدا رشته ای برای شروع برنامه برای کاربر نمایش داده می‌شود . بدین صورت که در داخل v0 مقدار ۴ را قرار می‌دهیم (به معنای print string) و در ادامه ادرس رشته مورد نظر را در a0 قرار می‌دهیم . برای ورودی گرفتن رشته ای از پرانتز های مورد نظر ابتدا در v0 مقدار ۸ را قرار می‌دهیم (به معنای ورودی گرفتن رشته) و ادرس مکان حافظه که قرار است رشته مورد نظر نگه داشته شود به a0 پاس داده می‌شود . counter برنامه که قرار است به روی رشته پیمایش کند ابتدا با دستور li به 0 مقداردهی می‌شود . و انتها ادرس حلقه برای پیمایش هم از طریق دستور add حساب می‌شود .

```
main:
    li $v0, 4
    la $a0, prompt
    syscall # print prmot
    li $v0, 8
    la $a0, buffer
    li $a1, 256 # maximum string that you can read
    syscall # get input
    li $s0 , 0 #counter
    la $t0 , buffer #get the address of buffer
    addi $t1 , $t0 , 256
```

بخش حلقه برنامه بدین صورت است که بر روی تک تک رشته ۲۵۶ بایتی پیمایش میکنیم و به ازای هر پرانتز کانتر را به علاوه یک کرده و به ازای پرانتز بسته منها یک و درصورتی که در انتها تک پیمایش مقدار منفی برای کانتر وجود داشت در همانجا مشخص میکنیم که برنامه NO را پرینت کند در غیراینصورت تمام ۲۵۶ بایت را پیمایش کند .

```
loop :
    beq $t0 , $t1  , endmain
    lb $t2 , 0($t0)
    bne $t2 , '(' , open
    addi $s0 , $s0 , 1
    open:
    bne $t2 , ')' , close
    addi $s0 ,$s0 , -1
    close:
    bltz $s0 , endmain
    addi $t0 , $t0 , 1
    j loop
```

```
endmain :
    beqz $s0 , printYes
    li $v0, 4
    la $a0, No
    syscall # print No
    j end
printYes :
    li $v0, 4
    la $a0, Yes
    syscall # print Yes

end :
    li $v0, 10
    syscall
```

تست برنامه :

```
enter your parentheses
(())()
Yes
-- program is finished running --
```

```
enter your parentheses
(((()))))
No
-- program is finished running --
```

2)

در بخش main برنامه ابتدا مقدار دهی res را انجام می‌دهیم سپس به استفاده از syscall عددی که به عنوان ارگومان تابع پاس داده می‌شود را می‌گیریم . سپس ارگومان های تابع را از a0 تا a2 قرار می‌دهیم

```
main:
        li $s0 , 0 # res = 0
        li $v0, 5 # getting n
        syscall
        move $s1, $v0 # set n
        addi $a1 , $s1 , 0 #passing n to a1
        li $a0 , 1
        li $a2 , 1

        jal func # calling function ans save return address

        li $v0, 1
        move $a0 , $s0 # print res
        syscall
        li $v0, 10  # finishing programme
        syscall
```

در بخش پیاده سازی function ابتدا ارگومان های پاس داده شده توسط a0 تا a2 را نگه می‌داریم و بعد از آن شرط اولیه برنامه که n برابر با 0 باشد را چک میکنیم تا اگر شرط برقرار بود به مقدار res یکی اضافه کنیم

```
func:
        # saving argumants in S2,S3,S4
        move $s2 , $a0  # i
        move $s3 , $a1  # n
        move $s4 , $a2  # index
        addi $s5 ,$s2 , 0 # j = i
        bnez $s3 , loop # if n not equal to zero
                addi $s0 , $s0 , 1 # if n == 0 res ++
```

در بخش دوم تابع که بخش بازگشتی آن هم در اینجا پیاده شده است بدین صورت است که ابتدا شرط اصلی برنامه چک می‌شود و در صورتی که شرط نقض شده باشد تابع صدا زده شده تمام می‌شود و در بخش endfucntion صرفا از دستور jr برای برگشت به instruction بعدی برنامه قبل از صدا زدن pc استفاده می‌شود . درغیراینصورت باید تابع را به صورت بازگشتی صدا بزنیم و ارگومان های مورد نظر را دوباره درون a0 تا a2 قرار دهیم و پس از آن قبل از صدا زدن تابع بازگشتی متغیر های حال حاضر تابع بازگشتی را نگه داریم بدین صورت که از sp استفاده میکنیم تا مقدار ها درون stack نگه داری شود ، روش کاری بدین صورت است که ابتدا۴ بایت sp را کم می کنیم تا فضا برای قرار دادن متغیر مورد نظر فراهم شود و سپس آن در در stack قرار می‌دهیم تا بتوانیم تابع را فراخوانی کنیم

```
loop:
        bgt $s5 , $s3 , endfunc # if j > n endfunction
        add $a0 , $zero , $s5 # a0 = j
        sub $a1 , $s3 , $s5 # a1 = n - j
        addi $a2 , $s4 , 1 # a2 = index + 1

        sub $sp,$sp,4 # allocating space in stack
        sw $s5,($sp)

        sub $sp,$sp,4
        sw $s2,($sp)

        sub $sp,$sp,4
        sw $s3,($sp)

        sub $sp,$sp,4
        sw $s4,($sp)

        sub $sp,$sp,4
        sw $ra,($sp)

        jal func
```

در ادامه کد هم مقدار های تابع هنگامی که تابع درونی به پایان رسیده است را با استفاده از stack و این سری با ۴ بایت ۴ بایت اضافه کردن می توانیم به متغیر های قبل از صدا زدن تابع درونی دسترسی داشته باشیم

```
        jal func

        lw $ra,($sp)
        addiu $sp,$sp,4

        lw $s4,($sp)
        addiu $sp,$sp,4

        lw $s3,($sp)
        addiu $sp,$sp,4

        lw $s2,($sp)
        addiu $sp,$sp,4

        lw $s5,($sp)
        addiu $sp,$sp,4

        addi $s5 , $s5 , 1
        j loop
```

```
4
5
-- program is finished running --

22
1002
-- program is finished running --
```

3)

در این بخش الگوریتم bubble-sort پیاده سازی می‌شود. در data segment ابتدا یک آرایه به طول ۲۵۶ بایت نگه داری می‌شود تا با گرفتن اعداد ورودی آن ها را در آرایه نگه داریم و همینطور دو متغیر از نوع asciiz داریم تا برای پرینت از آن ها استفاده شوند . در بخش main ابتدا طول آرایه را ورودی میگیریم سپس ابتدا آدرس آرایه را در t1 نگه می‌داریم

```
.data
array : .space 256
newline : .asciiz  " \0"
test : .asciiz "-\0"
.text
main :
        li $v0 , 5
        syscall
        move $s0 , $v0  # n
        li $t0 , 0
        la $t1 , array
```

در بخش inputloop ابتدا چک میکنیم که به آخرین عدد رسیده ایم یا نه که در غیراینصورت متغیر i که یکی یکی جلو می‌رود را ۲ تا به چپ شیفت می‌دهیم تا به آدرس ایندکس آرایه اشاره کند . سپس عدد را ورودی گرفته و در آدرس ذکر شده عدد را save می‌کنیم

```
inputloop :
        beq $t0 , $s0 , endinputloop
        sll $t3 , $t0 , 2
        add $t2 , $t1 , $t3
        li $v0 , 5
        syscall
        sw $v0 , 0($t2)
        addi $t0 , $t0 , 1
        j inputloop
```

بخش اصلی الگوریتم بدین صورت است که ابتدا از ۰ تا n-1 بار حلقه خارجی تکرار می‌شود و حلقه داخلی از ۰ تا n-1-i بار تکرار می‌شود و هر بار که عدد j+1 بزرگتر از j باشد این دو را swap می‌کند .

```
outerloop:
        beq $t0 , $t2 , endouterloop # i < n-1
        li $t3 , 0 # j = 0
        sub $t4 , $t2 , $t0 # t4 = (n-1) - i

innerloop:
        beq $t3 , $t4 , endinnerloop

        add $t5 , $t3 , $zero
        sll $t5 , $t5 , 2 # j << 2
        add $t5 , $t5 , $t1 # add j with base address of array
        lw $t6 , 0($t5)
        lw $t7 , 4($t5)

        bge $t7 , $t6 , smaller
        sw $t6 , 4($t5)
        sw $t7 , 0($t5)
```

اگر دو عدد ترتیب صعودی را رعایت کرده بودند صرفا j را یکی اضافه می‌کنیم و اگر حلقه داخلی به اتمام برسد i را یکی اضافه میکنیم .

```
smaller:
        addi $t3 , $t3 , 1 # j++
        j innerloop
endinnerloop:
        addi $t0 , $t0 , 1 # i++
        j outerloop
endouterloop:
        li $t0 , 0
        la $t1 , array
```

خروجی :

```
5
8
3
19
1
1
1 1 3 8 19
```