Práctica

1

1



**Objetivo**

Seleccionar las instrucciones de control de flujo del programa adecuadas en la manipulación de cadenas, para desarrollar aplicaciones de sistemas basados en microprocesador, mediante el análisis de su funcionalidad, de forma responsable y eficiente.

**Desarrollo**

1. Cree un programa llamado **P11.asm** que contenga la siguiente rutina:

**substr**: almacena en una cadena una copia de una porción de otra cadena. Recibe en ESI la dirección de una cadena fuente, en EDI la dirección de la cadena destino, en BX la posición inicial a copiar y en CX la cantidad de caracteres.

Si la cadena es más corta que los caracteres solicitados en CX, el procedimiento copia todos los posibles. Si la posición en BX es mayor que la longitud de la cadena, el procedimiento retorna un -1 en EAX, de lo contrario retorna 0.

Ejemplo: mov esi, cadena ; si lá cádená es “Holá mundo” mov edi, destino

mov bx, 1 ; copiar a partir de la posición 1 mov cx, 5 ; copiar 5 caracteres

call substr ; destino es “olá m”, retorná EAX = 0

## Código

section .data

   source: db "ABCDEFGHIJKLMNOPQRSTUVWXYZ",0

   error: db "hay errores en los tamaños"

   error\_L: equ $-error

   destiny: db "1234567890",0

section .bss

section .text

global \_start:

\_start:

   mov esi,source

   mov edi,destiny

   mov ebx,4

   mov ecx,4

   call substr

   cmp eax,-1

   je .sizeError

   pushad

   mov eax, 4   ;servicio

   mov ebx, 1   ;Entrada

   mov ecx, destiny ;Cadena

   mov edx, 10  ;Caracteres

   int 80h

   popad

   jmp .end

.sizeError:

   pushad

   mov eax, 4   ;servicio

   mov ebx, 1   ;Entrada

   mov ecx, error ;Cadena

   mov edx, error\_L  ;Caracteres

   int 80h

   popad

.end:

   ;End program

   mov eax,1

   mov ebx,0

   int 80h

substr:

   pushad

   call testStr ;Verificamos los tamaños de ambas cadenas

   cmp eax,-1

   je .sizeError

   xchg esi,ebx

.cicle:

   mov al,[ebx+esi]

   mov [edi],al

   inc edi

   inc esi

   loop .cicle

   popad

   mov eax,0

   jmp .end

.sizeError:

   popad

   mov eax,-1

.end:

   ret

testStr:

   pushad

   push ecx

   push edi

   mov edi,esi

   call getLenght ;obtiene el tamaño de la cadena en edi

   cmp eax,ecx

   jle .sizeErrorAndPopReg

   add ecx,ebx

   cmp eax,ecx

   jl .sizeErrorAndPopReg

   pop edi

   pop ecx

   call getLenght

   cmp eax,ecx

   jle .sizeError

   cmp eax,ecx

   jae .endTest

.sizeErrorAndPopReg:

   pop edi

   pop ecx

.sizeError:

   popad

   mov eax,-1

   jmp .endError

.endTest:

   popad

   mov eax,0

.endError:

   ret

;Obtiene el tamaño de una cadena (Asusmiendo que termina en 0x0)

getLenght:

   push ecx

   push ebx

   push edi

   mov ebx,edi

   mov eax,0 ;caracter a buscar

   mov ecx,100

   cld

   repne scasb ;repne = repetir mientras no sea igual

   ;scasb = buscar un caracter (byte \*al\*) en una cadena (ebx)

   sub edi,ebx ;Obtenemos la diferencia

   mov eax,edi

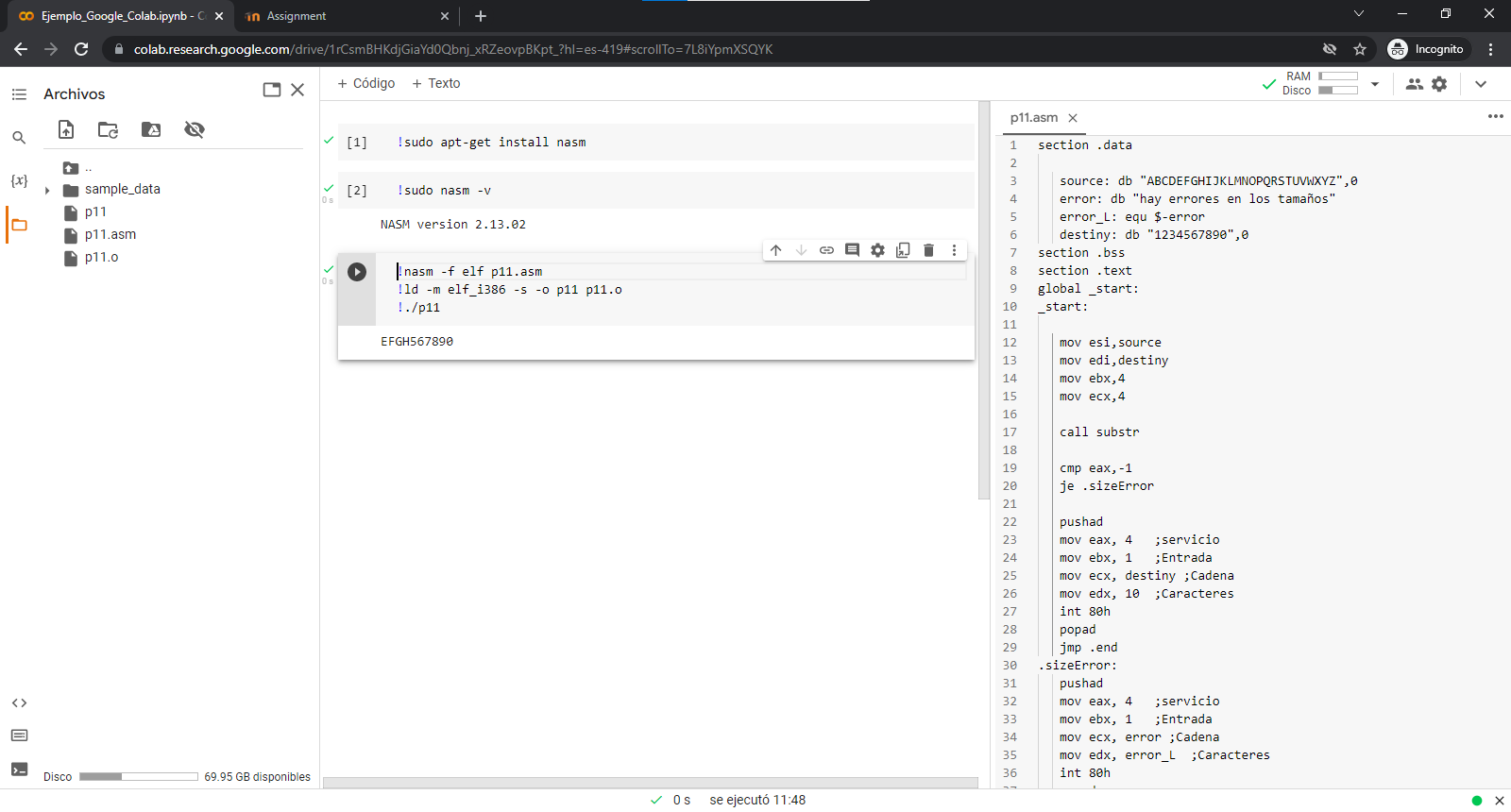
   pop edi

   pop ebx

   pop ecx

   ret

## Resultado en línea de comandos



## Conclusiones y comentarios

Aprender el manejo de cadenas, así como el direccionamiento es uno de los puntos más importantes por aprender a la hora de codificar un programa, ya que puede evitar un gran número de problemas que surgen al manipular direcciones de memoria sin el entendimiento adecuado de como funciona.

## Dificultades en el desarrollo

Mi mayor problema al momento de codificar fue el uso de la pila, existieron momentos en los cuales empujaba registros a la pila y por algún motivo ahí los mantenía, así que cuando utilizaba la instrucción “ret” crasheaba el programa.

## References

Assembly - SCAS Instruction. (2022). Retrieved 2 May 2022, from https://www.tutorialspoint.com/assembly\_programming/assembly\_scas\_instruction.htm

REPNE - Repeat String Operation Prefix. (2022). Retrieved 2 May 2022, from https://faydoc.tripod.com/cpu/repne.htm