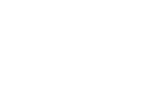


Prác

tica



9



# Objetivo

Seleccionar las instrucciones de control de flujo del programa adecuadas, 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 **P9.asm** que contenga las siguientes rutinas:

a) **printNumBase**: imprime el número en EAX en el formato según la base dada en el registro BL. La representación del número en caracteres ASCII además de desplegarse en pantalla, también se almacena en una cadena en memoria apuntada por ESI.

Ejemplos:

mov eax,AB385h

mov bl, 10 ; desplegar el valor en decimal mov

esi, cadena call printNumBase ; imprime y almacena en *cadena* “1000807”

mov eax,6A4B9D01h

mov bl, 16 ; desplegar el valor en hexadecimal mov esi, cadena call printNumBase ; imprime y almacena en *cadena* “10592FA”

mov eax, 47845384

mov bl, 12 ; desplegar el valor base 12 mov esi, cadena call printNumBase ; imprime y almacena en *cadena* “14034374”

mov eax, 2531DEh

mov bl, 2 ; desplegar el valor en binario mov esi, cadena call printNumBase ; imprime y almacena en *cadena* “1001010011000111011110”

En el código anterior se ejemplifica la impresión del registro EAX en decimal, hexadecimal, base12 y binario, sin embargo, el procedimiento debe ser **funcional para cualquier base solicitada** que sea imprimible de acuerdo al límite de caracteres en la tabla ASCII. El procedimiento debe ser **genérico**, no realice invocaciones a **printBin** o **printHex**, haga la conversión por medio de divisiones.

1. **SetBit:** activa un bit del registro EAX. El número de bit a activar está dado por CL.
2. **ClearBit:**  desactiva un bit del registro EAX. El número de bit a desactivar está dado por CL.
3. **NotBit:**  invierte un bit del registro EAX. El número de bit a invertir está dado por CL.
4. **TestBit:**  copia un bit del registro EAX a la bandera de acarreo. El bit a copiar está dado por CL.
5. **Func1:** ingresan una cadena y obtienen:

**Ejemplo:** cad1 = “Hola Mundo” e imprimen “odnuM aloH” y se guarda en cadena2

mov esi, cad1 ;variable con la cadena

mov edi, cadena2 ;almacena la salida de func1 en cadena2 call func1 ;imprime “odnuM aloH”

1. **Func2:** ingresan un número menor a 9 y les regresa la multiplicación

mov eax, 8 ;valor desde el cual iniciara la función mov esi, cadena3 ;almacena la salida de func2 en cadena3 call func2 ;imprime 8\*7\*6\*5\*4\*3\*2\*1=40320

Validar que sí se hayan realizado los cambios, ya sea en la cadena, en el registro o en la bandera de acarreo.

# Codigo

section .data

   NL: db 13, 10

   NL\_L: equ $-NL

   ClauseText: db 10,13,"Inciso "

   ClauseText\_L: equ $-ClauseText

   ClauseLetter: db "a)",10,13

   ClauseLetter\_L: equ $-ClauseLetter

   Zero: db "El bit es cero",10,13

   Zero\_L: equ $-Zero

   One: db "El bit es uno",10,13

   One\_L: equ $-One

   Cad1: db "Hello World"

   Cad1\_L: equ $-Cad1

   char: db 'h-'

section .bss

   Cad resb 8

   temp resb 8

   Cad2 resb 8

   Cad2\_L resb 8

   Cad3 resb 64

   Cad3\_L resb 8

section .text

global \_start:

\_start:

   call printClause

   mov eax,0AB385h

   mov bl,10

   mov esi,Cad

   call printOriginalValue

   call printNumBase

   call printNl

   mov eax,6A4B9D01h

   mov bl,16

   mov esi,Cad

   call printOriginalValue

   call printNumBase

   call printNl

   mov eax,47845384h

   mov bl,12

   mov esi,Cad

   call printOriginalValue

   call printNumBase

   call printNl

   mov eax,2531D3h

   mov bl,2

   mov esi,Cad

   call printOriginalValue

   call printNumBase

   call printNl

   mov byte[char],10

   mov byte[char+1],13

   mov eax,0Fh

   call printClause

   call printOriginalValue

   call printNl

   mov cl,5

   call setBit

   call printOriginalValue

   call printNl

   call printClause

   mov cl,3

   call clearBit

   call printOriginalValue

   call printNl

   call printClause

   mov cl,1

   call notBit

   call printOriginalValue

   call printNl

   call printClause

   mov cl,2

   call testBit

   jc .carryOne

   mov eax, 4  ;Servicio

   mov ebx, 1  ;Salida

   mov ecx, Zero

   mov edx, Zero\_L

   int 80h

   jmp .next

.carryOne:   mov eax, 4  ;Servicio

   mov ebx, 1  ;Salida

   mov ecx, One

   mov edx, One\_L

   int 80h

.next:

   call printClause

   mov esi, Cad1

   mov edi, Cad2

   call Func1

   call printNl

   call printClause

   mov eax,6

   mov esi,Cad3

   call Func2

   ;End program

   mov eax,1

   mov ebx,0

   int 80h

Func1:

  pushad

  mov ecx,Cad1\_L

  mov [Cad2\_L],ecx

  add esi,ecx

.rev:

  mov al,[esi-1]

  mov [edi],al

  inc edi

  dec esi

  loop .rev

  sub edi,ecx

  mov eax, 4  ;Servicio

  mov ebx, 1  ;Salida

  mov ecx, Cad2

  mov edx, Cad1\_L

  int 80h

  popad

  ret

Func2:

   cmp eax,10

   jae .exception

   mov ecx,eax

   mov eax,1

   mov esi,0

   mov ebx,ecx

   add bl,'0'

   mov [Cad3+esi],bl

   inc esi

.factorial:

   mul cx

   dec cl

   jz .endFact

   mov bl,'\*'

   mov [Cad3+esi],bl

   inc esi

   mov ebx,ecx

   add bl,'0'

   mov [Cad3+esi],bl

   inc esi

   jmp .factorial

.endFact:

   mov bl,'='

   mov [Cad3+esi],bl

   inc esi

   mov [Cad3\_L],esi

   pushad

   mov eax, 4  ;Servicio

   mov ebx, 1  ;Salida

   mov ecx, Cad3

   mov edx, esi

   int 80h

   popad

   mov bl,10

   mov esi,Cad

   call printNumBase

.exception ret

printNumBase:

   pushad

   mov edi,0

   mov bh,0

   mov ecx,0

.cicle:   mov edx,0

   div ebx

   cmp dl,10d

   jb .direct

   add dl,7

.direct add dl,'0'

   mov [temp+edi],dl

   inc edi

   inc cl

   cmp eax,0

   jnz .cicle

   mov ebx,esi

   mov esi,0

   push ecx

.revNum:

   mov al,[temp+ecx-1]

   mov [ebx+esi],al

   inc esi

   loop .revNum

   ;Imprimir cadena

   mov eax, 4  ;Servicio

   mov ecx, ebx

   mov ebx, 1  ;Salida

   pop edx

   int 80h

   call printNl

   popad

   ret

setBit:

   ror eax,cl

   or eax,1

   rol eax,cl

   ret

clearBit:

   ror eax,cl

   and eax,0FFFFFFFEh

   rol eax,cl

   ret

notBit:

   ror eax,cl

   xor eax,1

   rol eax,cl

   ret

testBit:

   push eax

   ror eax,cl

   xor eax,0FFFFFFFEh

   rcr eax,1

   pop eax

   ret

printNl:

   pushad

   mov eax, 4

   mov ebx, 1

   mov ecx, NL

   mov edx, NL\_L

   int 80h

   popad

   ret

printClause:

   pushad

   ;Imprimir "Inciso "

   mov eax, 4  ;Servicio

   mov ebx, 1  ;Salida

   mov ecx, ClauseText

   mov edx, ClauseText\_L

   int 80h

   ;Imprimir "x)"

   mov eax, 4  ;Servicio

   mov ebx, 1  ;Salida

   mov ecx, ClauseLetter

   mov edx, ClauseLetter\_L

   int 80h

   add byte [ClauseLetter],1

   popad

   ret

printOriginalValue:

   pushad

   mov esi,Cad

   mov edx, eax

   mov ebx, 0fh

   mov cl, 28

.nxt: shr eax,cl

.msk: and eax,ebx

   cmp al, 9

   jbe .menor

   add al,7

.menor:add al,'0'

   mov byte [esi],al

   inc esi

   mov eax, edx

   cmp cl, 0

   je .print

   sub cl, 4

   cmp cl, 0

   ja .nxt

   je .msk

.print:

   mov eax, 4

   mov ebx, 1

   sub esi, 8

   mov ecx, esi

   mov edx, 8

   int 80h

   mov byte[char],'h'

   mov eax,4

   mov ebx,1

   mov ecx,char

   mov edx,2

   int 80h

   popad

   ret

# Conclusiones y comentarios

Es bastante útil el poder generalizar funciones ya que, gracias a esto podemos realizar diferentes procedimientos con una sola función. Como en esta práctica fue la conversión, en vez de ser necesario crear una función por cada base, se puede crear una sola que dependiendo de los datos introducidos haga la conversión a la base deseada.

# Dificultades en el desarrollo

Hacer la conversión y el manejo de cadenas es lo más complicado dentro de toda la práctica, sin embargo, en cuanto al manejo de cadenas con la práctica se hace cada vez más sencillo, ya que se logra entender su funcionamiento de mejor manera.