**Matricula: 20180999 Nombre: Junior Hernández**

Análisis de código generado por el compilador.

El objetivo de la presente tarea es crear un código en lenguaje C con las sgtes. Características.

1. Debe de contener al menos una función con paso de parámetros de entrada.
2. Debe de contener una función que retorne un valor.
3. Usted elige la operación  que va realizar por si programa, es decir, que usted debe de elegir una idea de la tarea a realizar por su programa. Considerando que tengan más de 30 líneas de código.
4. Del programa en cuestión usted debe de comentarlo línea por línea , y en lenguaje C.
5. Luego debe de desensamblar el código en C a assembler  y comentarlo línea por línea, indicando que se consigue con cada instrucción.
6. Debe de elaborar un reporte donde explique cómo funciona el Paso de parámetros en assembler.
7. Compile de dos formas, a-) con optimizaciones, b-)  sin optimización. Evalúe ambos códigos, y explique las diferencias de uno y el otro.

**Programa en C:**

#include <stdio.h> //Aqui se llama la libreria stdio

#include <stdlib.h> //aqui se llama la libreria stdlib

int cambiarelorden (int); //Aqui establecemos la estructura de la funcion para cambiar el orden

int main() //AQui va el main

{

int num1, numvolt; //Aqui establecemos dos variables integer un representando el numero y el otro el numero volteado

char deci; //Aqui la variable que acumula a c si el usuario quiere continuar

do // Establecemos un bucle

{

printf("Ingrese n%cmero: \n", 163); //Mensaje para que ingrese el numero

scanf("%d",&num1); //Para leer y guardar lo que ingresa el usuario por el teclado

if(num1 < 0) //Decision para saber si el numero es menor que 0

{

printf("El valor debe ser mayor que 0.\n"); //Muestra el mensaje de que el numero debe ser mayor que 0

printf("Presione C si desea continuar. De lo contrario presione cualquier tecla. \n"); //Mensaje para que el usaurio vuelva a ingresar una C

scanf(" %c",&deci); //Lee y guarda lo ingresado por el usuario desde el teclado

system("cls"); //Se cierra/limpia la pantlla de la consola

}

else

{

if ( num1 < 10 && num1 > 0) //Decision para saber si el numero es mayor que 0 y menor que 10

{

printf("El valor debe ser mayor que 10.\n"); //Mensaje aclarando que debe de ingresar un numero mayor que 10

printf("Presione C si desea continuar. De lo contrario presione cualquier tecla. \n");

scanf(" %c",&deci);

system("cls"); //Se cierra/limpia la pantlla de la consola

}

else //Si no se cumple que el numero es menor a 10

{

numvolt = cambiarelorden(num1); //Llamamos la funcion de cambiar orden para voltear el numero y lo guardamos en la variable numvolt

if (numvolt == num1) // Si el numero volteado es igual que el numero sin voltear

{

printf("%d: es pal%cndromo \n", num1, 161); //Es palindromo

}

else

{

printf("%d: no es pal%cndromo \n", num1, 161); //No es palindromo

}

printf("Presione C si desea continuar. De lo contrario presione cualquier tecla. \n");

scanf(" %c",&deci);

system("cls");

}

}

}

while(deci == 'c' || deci == 'C'); //El bucle continuara mientras el usuario siga ingresando una C

}

int cambiarelorden(int num) //Se establece la funcion cambiarelorden tomando como paremetro un numero

{

int digi, numvolte=0;

while(num != 0) //Mientras el numero sea diferente a 0

{

digi=num%10; //SAcamos el mod 10 del parametro para coger un digito individual del mismo

numvolte=numvolte\*10+digi; //Guardamos el numero volteado agregando cada digito

num/=10; //VAmos dividiendo en 10 el parametro

}

return numvolte; //Retorna el numero volteado

}

**Programa traducido a Assembly:**

; Disassembly of file: t6enC.o

; Tue Nov 2 19:54:42 2021

; Type: ELF64

; Syntax: NASM

; Instruction set: Pentium Pro, x64

default rel

global cambiarelorden: ; establecemos la funcion de cambiarelorden eque habiamos hecho en el programa en c

global main: ; establecemos el main

extern \_\_stack\_chk\_fail ; terminar una función en caso de desbordamiento de pila

extern \_\_printf\_chk ; formatear e imprimir datos, con comprobación de pila

extern system ; near

extern \_\_isoc99\_scanf ; Para el cumplimiento estricto de ISO C99 o POSIX, no permita% as,% aS y% a [extensión GNU que entra en conflicto con% a válido seguido de letras, S o].

extern puts ; Cuando el código está vinculado, se conoce la dirección de put () la función el código lo tratará como una variable entera

extern GLOBAL\_OFFSET\_TABLE ; Vive en una sección del archivo ELF que ha sido llamada. El área reservada para estas direcciones.

SECTION .text ; sección número 1, código

cambiarelorden:; Comienzo de la función

xor r8d, r8d ; Realiza una operación lógica XOR bit a bit entre el destino y la fuente

test edi, edi ; Comparación lógica de los dos operandos; hace una 'y' lógica sin guardar el resultado.

jz ?\_002 ; salta a la etiqueta ?\_002 si igual.

; Filling space: 5H

; Filler type: Multi-byte NOP

; db 0FH, 1FH, 44H, 00H, 00H

ALIGN 8

?\_001: movsxd rax, edi ; se utiliza para copiar un valor de 32 bits haciendo la extensión sobre un valor de 64 bits.

mov edx, edi ; instrucción genérica para mover un dato desde un origen a un destino.

lea ecx, [r8+r8\*4] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

imul rax, rax, 1717986919 ; ejecuta una multiplicación firmada de un byte.

sar edx, 31 ; realiza un desplazamiento aritmético a la derecha en el operando de destino.

sar rax, 34 ; realiza un desplazamiento aritmético a la derecha en el operando de destino.

sub eax, edx ; resta aritmética de los dos operandos.

lea edx, [rax+rax\*4] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

add edx, edx ; suma aritmética de los dos operandos.

sub edi, edx ; resta aritmética de los dos operandos.

lea r8d, [rdi+rcx\*2] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

mov edi, eax ; instrucción genérica para mover un dato desde un origen a un destino.

test eax, eax ; Comparación lógica de los dos operandos; hace una 'y' lógica sin guardar el resultado.

jnz ?\_001 ; salta a la etiqueta si diferente.

?\_002: mov eax, r8d ; instrucción genérica para mover un dato desde un origen a un destino.

ret ; retorno de subrutina.

; aqui termina la funcion cambiarelorden

SECTION .data ; section number 2, data

SECTION .bss ; section number 3, bss

SECTION .rodata.str1.1 ; section number 4, const

; Aqui se guardan y representan los mensajes y variables que tenemos en el programa en C recordando que db define una variable de tipo byte, 8 bits.

LC0: ; byte

db 49H, 6EH, 67H, 72H, 65H, 73H, 65H, 20H ; 0000 \_ Ingrese

db 6EH, 25H, 63H, 6DH, 65H, 72H, 6FH, 3AH ; 0008 \_ n%cmero:

db 20H, 0AH, 00H ; 0010 \_ ..

LC1: ; byte

db 25H, 64H, 00H ; 0013 \_ %d.

LC4: ; byte

db 20H, 25H, 63H, 00H ; 0016 \_ %c.

LC5: ; byte

db 63H, 6CH, 73H, 00H ; 001A \_ cls.

LC7: ; byte

db 25H, 64H, 3AH, 20H, 65H, 73H, 20H, 70H ; 001E \_ %d: es p

db 61H, 6CH, 25H, 63H, 6EH, 64H, 72H, 6FH ; 0026 \_ al%cndro

db 6DH, 6FH, 20H, 0AH, 00H ; 002E \_ mo ..

LC8: ; byte

db 25H, 64H, 3AH, 20H, 6EH, 6FH, 20H, 65H ; 0033 \_ %d: no e

db 73H, 20H, 70H, 61H, 6CH, 25H, 63H, 6EH ; 003B \_ s pal%cn

db 64H, 72H, 6FH, 6DH, 6FH, 20H, 0AH, 00H ; 0043 \_ dromo ..

SECTION .rodata.str1.8 ; section number 5, const

LC2: ; byte

db 45H, 6CH, 20H, 76H, 61H, 6CH, 6FH, 72H ; 0000 \_ El valor

db 20H, 64H, 65H, 62H, 65H, 20H, 73H, 65H ; 0008 \_ debe se

db 72H, 20H, 6DH, 61H, 79H, 6FH, 72H, 20H ; 0010 \_ r mayor

db 71H, 75H, 65H, 20H, 30H, 2EH, 00H, 00H ; 0018 \_ que 0...

LC3: ; byte

db 50H, 72H, 65H, 73H, 69H, 6FH, 6EH, 65H ; 0020 \_ Presione

db 20H, 43H, 20H, 73H, 69H, 20H, 64H, 65H ; 0028 \_ C si de

db 73H, 65H, 61H, 20H, 63H, 6FH, 6EH, 74H ; 0030 \_ sea cont

db 69H, 6EH, 75H, 61H, 72H, 2EH, 20H, 44H ; 0038 \_ inuar. D

db 65H, 20H, 6CH, 6FH, 20H, 63H, 6FH, 6EH ; 0040 \_ e lo con

db 74H, 72H, 61H, 72H, 69H, 6FH, 20H, 70H ; 0048 \_ trario p

db 72H, 65H, 73H, 69H, 6FH, 6EH, 65H, 20H ; 0050 \_ resione

db 63H, 75H, 61H, 6CH, 71H, 75H, 69H, 65H ; 0058 \_ cualquie

db 72H, 20H, 74H, 65H, 63H, 6CH, 61H, 2EH ; 0060 \_ r tecla.

db 20H, 00H, 00H, 00H, 00H, 00H, 00H, 00H ; 0068 \_ .......

LC6: ; byte

db 45H, 6CH, 20H, 76H, 61H, 6CH, 6FH, 72H ; 0070 \_ El valor

db 20H, 64H, 65H, 62H, 65H, 20H, 73H, 65H ; 0078 \_ debe se

db 72H, 20H, 6DH, 61H, 79H, 6FH, 72H, 20H ; 0080 \_ r mayor

db 71H, 75H, 65H, 20H, 31H, 30H, 2EH, 00H ; 0088 \_ que 10..

SECTION .text.startup ; section number 6, code

main: ; inicio de la funcion main

push r12 ; instrucción que mueve el operando de la instrucción a la cima de la pila.

push rbp ; instrucción que mueve el operando de la instrucción a la cima de la pila.

push rbx ; instrucción que mueve el operando de la instrucción a la cima de la pila.

mov ebx, 3435973837 ; instrucción genérica para mover un dato desde un origen a un destino.

sub rsp, 16 ; resta aritmética de los dos operandos.

;La operación push copia un valor a la pila tal cual. No tiene mas. No enlaza,

;ni crea referencias, ni nada mágico.

;Realiza una copia, por lo que ambos elementos (el original y el copiado) son totalmente independientes.

;Las modificaciones en uno no afectan para nada al otro.

;Por lo tanto, si haces un push y luego modificas el valor, dichos cambios no se trasladan a la pila.

; Nota: la dirección no es relativa a la copia

; Nota: dirección de memoria absoluta sin reubicación

mov rax, qword [fs:abs 28H] ; instrucción genérica para mover un dato desde un origen a un destino.

mov qword [rsp+8H], rax ; instrucción genérica para mover un dato desde un origen a un destino. Nota: qword es un especificador de tamaño valido que posiciona en memoria de 64 bits

xor eax, eax ; Realiza una operación lógica XOR bit a bit entre el destino y la fuente.

lea rbp, [rsp+4H] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

lea r12, [rsp+3H] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

jmp ?\_005 ; salto a la funcion ?\_005

; Filling space: 3H

; Filler type: Multi-byte NOP

; db 0FH, 1FH, 00H

ALIGN 8

func1: lea rdi, [rel LC6] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

call puts ; llama a puts

?\_004: lea rdi, [rel LC3] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

call puts ; vuelve a llamar a puts

mov rsi, r12 ; instrucción genérica para mover un dato desde un origen a un destino.

lea rdi, [rel LC4] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

xor eax, eax ; Realiza una operación lógica XOR bit a bit entre el destino y la fuente.

call \_\_isoc99\_scanf ; llama al lector

lea rdi, [rel LC5] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

call system ; hace llamada al sistema

movzx eax, byte [rsp+3H] ; copia el valor del operando fuente sobre el operando destino sobrescribiendo el valor original del operando destino, añadiendo ceros delante.

and eax, 0FFFFFFDFH ; comparacion de una y logica.

cmp al, 67 ; compara dos parametros

jne ?\_008 ; salta a la etiqueta si diferente, si el bit de cero no es activo.

?\_005: mov edx, 163 ; instrucción genérica para mover un dato desde un origen a un destino.

lea rsi, [rel LC0] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

mov edi, 1 ; instrucción genérica para mover un dato desde un origen a un destino.

xor eax, eax ; Realiza una operación lógica XOR bit a bit entre el destino y la fuente.

call \_\_printf\_chk ; llamada a al funcion que le da formato a los print para c en este caso

mov rsi, rbp ; instrucción genérica para mover un dato desde un origen a un destino.

lea rdi, [rel LC1] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

xor eax, eax ; Realiza una operación lógica XOR bit a bit entre el destino y la fuente.

call \_\_isoc99\_scanf ; llama a la funcion del lector

mov r8d, dword [rsp+4H] ; instrucción genérica para mover un dato desde un origen a un destino.

test r8d, r8d ; Comparación lógica de los dos operandos; hace una 'y' lógica sin guardar el resultado.

js ?\_007 ; salta a la etiqueta si el bit de signo es activo.

lea eax, [r8-1H] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

cmp eax, 8 ; compara dos parametros

jbe func1 ; salta a la etiqueta si es más pequeño o igual.

mov edx, r8d ; instrucción genérica para mover un dato desde un origen a un destino.

xor ecx, ecx ; Realiza una operación lógica XOR bit a bit entre el destino y la fuente.

test r8d, r8d ; Comparación lógica de los dos operandos; hace una 'y' lógica sin guardar el resultado.

je ?\_009 ; salta a la etiqueta si igual.

; Filling space: 5H

; Filler type: Multi-byte NOP

; db 0FH, 1FH, 44H, 00H, 00H

ALIGN 8

?\_006: mov eax, edx ; instrucción genérica para mover un dato desde un origen a un destino.

lea esi, [rcx+rcx\*4] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

imul rax, rbx ; ejecuta una multiplicación firmada de un byte.

shr rax, 35 ; esplazamiento lógico a la derecha

lea ecx, [rax+rax\*4] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

add ecx, ecx ; suma aritmética de los dos operandos.

sub edx, ecx ; resta aritmética de los dos operandos.

lea ecx, [rdx+rsi\*2] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

mov edx, eax ; instrucción genérica para mover un dato desde un origen a un destino.

test eax, eax ; Comparación lógica de los dos operandos; hace una 'y' lógica sin guardar el resultado.

jnz ?\_006 ; salta a la etiqueta si diferente.

cmp r8d, ecx ; Comparación entre parametros.

jz ?\_009 ; salta a la etiqueta si igua

mov ecx, 161 ; instrucción genérica para mover un dato desde un origen a un destino.

mov edx, r8d ; instrucción genérica para mover un dato desde un origen a un destino.

mov edi, 1 ; instrucción genérica para mover un dato desde un origen a un destino.

xor eax, eax ; Realiza una operación lógica XOR bit a bit entre el destino y la fuente.

lea rsi, [rel LC8] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

call \_\_printf\_chk ; llama a la funcion print para dar formato

jmp ?\_004 ; salta a la etiqueta indicada

; Filling space: 6H

; Filler type: Multi-byte NOP

; db 66H, 0FH, 1FH, 44H, 00H, 00H

ALIGN 8

?\_007: lea rdi, [rel LC2] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

call puts ; llama a puts

jmp ?\_004 ; salta a la etiqueta indicada

; Filling space: 7H

; Filler type: Multi-byte NOP

; db 0FH, 1FH, 80H, 00H, 00H, 00H, 00H

ALIGN 8

?\_008: mov rax, qword [rsp+8H] ; instrucción genérica para mover un dato desde un origen a un destino.

; Note: Address is not rip-relative

; Note: Absolute memory address without relocation

xor rax, qword [fs:abs 28H] ; Realiza una operación lógica XOR bit a bit entre el destino y la fuente.

jnz ?\_010 ; salta a la etiqueta si diferente.

add rsp, 16 ; suma aritmética de los dos operandos.

xor eax, eax ; Realiza una operación lógica XOR bit a bit entre el destino y la fuente.

pop rbx ; mueve el dato que se encuentra en la cima de la pila al operando destino.

pop rbp ; mueve el dato que se encuentra en la cima de la pila al operando destino.

pop r12 ; mueve el dato que se encuentra en la cima de la pila al operando destino.

ret ; retorno de subrutina.

; Filling space: 5H

; Filler type: Multi-byte NOP

; db 0FH, 1FH, 44H, 00H, 00H

ALIGN 8

?\_009: mov ecx, 161 ; instrucción genérica para mover un dato desde un origen a un destino.

mov edx, r8d ; instrucción genérica para mover un dato desde un origen a un destino.

mov edi, 1 ; instrucción genérica para mover un dato desde un origen a un destino.

xor eax, eax ; Realiza una operación lógica XOR bit a bit entre el destino y la fuente.

lea rsi, [rel LC7] ; carga la dirección de la referencia de ubicación por el operando de origen al operando de destino.

call \_\_printf\_chk ; llama a la funcion print para dar formato

jmp ?\_004 ; salto a la etiqueta indicada

; main End of function

?\_010: ; Local function

; Note: Function does not end with ret or jmp

call \_\_stack\_chk\_fail ; llamada de la funcion para la terminacion por desbordamiento de la pila

; esto es para la llamada del compilador de C el GNU

SECTION .note.gnu.property ; section number 7, const

db 04H, 00H, 00H, 00H, 10H, 00H, 00H, 00H ; 0000 \_ ........

db 05H, 00H, 00H, 00H, 47H, 4EH, 55H, 00H ; 0008 \_ ....GNU.

db 02H, 00H, 00H, 0C0H, 04H, 00H, 00H, 00H ; 0010 \_ ........

db 03H, 00H, 00H, 00H, 00H, 00H, 00H, 00H ; 0018 \_ ........

**Salida:**

