Examen de Arquitectura de Computadoras

21 de febrero de 2022

Instrucciones:

- **Indique su nombre, apellido y número de cédula en todas las hojas que entregue.**
- **Escriba las hojas de un solo lado.**
- **Las hojas deben estar numeradas y en la primer hoja debe escribirse el total.**
- **Empiece cada ejercicio en una hoja nueva.**
- **Apague su celular. No puede utilizar material ni calculadora.**
- **La duración del examen es de tres horas. En dicho tiempo debe también completar sus datos.**
- **Solo se contestarán dudas de letra. No se aceptan preguntas en los últimos 30 minutos del examen.**
- **Para aprobar debe contestar correctamente un ejercicio entero y dos preguntas.**

Pregunta 1

Considere un CPU operando con una capacidad de direccionamiento de 16 MBytes, memoria direccionable de a byte, y una memoria caché de 32 líneas de 128 Bytes cada una, que utiliza una función de correspondencia asociativa por conjuntos de cuatro vías.
Indique cuál es el hit rate de ejecutar el siguiente fragmento de código, asumiendo que inicialmente la memoria caché se encuentra vacía. Justifique su respuesta.

```
LOAD 0xA64CDE, R1
LOAD 0xA64CFE, R2
MUL R1 R2
LOAD 0xA646FE, R3
MUL R1 R3
```

Nota: La operación `LOAD MEM R1` carga el contenido almacenado en la posición de memoria `MEM` en el registro `R1`. La operación `MUL R1 R2` multiplica el contenido del registro `R1` por el contenido del registro `R2` y almacena el resultado en el registro `R1`.

Pregunta 2

Considere una arquitectura de Von Neumann de 16 bits, registros de 16 bits y formato de instrucción:

CÓDIGO de OPERACIÓN (5 bits)	REGISTRO (3 bits)	DIRECCIÓN (8 bits)

Indique, fundamentando su respuesta:

a) ¿Cuántos registros como máximo dispone la arquitectura?

b) ¿Cuál es el máximo espacio de memoria direccionable utilizando direccionamiento directo?

c) ¿Cuál es el máximo espacio de memoria direccionable utilizando direccionamiento indirecto por memoria?

Pregunta 3

a) Dibuje el circuito interno de un decodificador de 3 bits.

b) Construya un decodificador de 3 bits a partir de un demultiplexor de 3 bits de control.

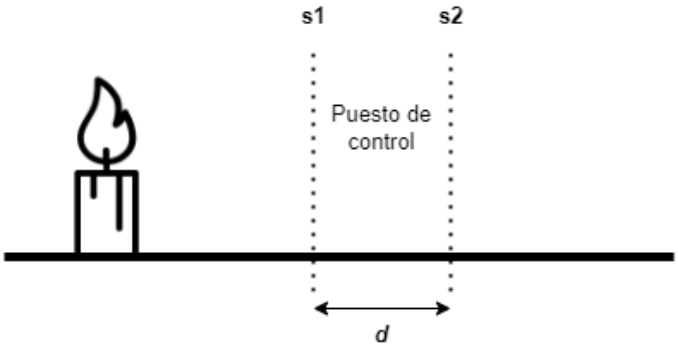
Pregunta 4

Implemente en assembler 8086 un procedimiento que cargue completamente el vector de interrupciones a partir de direcciones segmentadas recibidas desde E/S. En el puerto de E/S de tipo palabra, de solo lectura, en la dirección DATA, se obtienen en orden creciente los datos a cargar (para cada elemento del vector primero el desplazamiento y luego el segmento). El bit 3 del puerto de E/S, de tipo palabra, de solo lectura, en la dirección CTRL, indica cuando hay una palabra válida en DATA y se resetea al leerse dicho puerto.

Problema 1

La empresa **Olores de Mealy** fabrica velas aromáticas y en esta oportunidad le han encomendado a usted el diseño de un nuevo sistema automático de control del diámetro de las velas.

Las velas se mueven a través de una cinta transportadora y llegan al puesto de control con la cera caliente, en moldes que pueden ensancharse para permitir que las velas aumenten su diámetro. El objetivo del puesto de control es asegurar que todas las velas tengan un diámetro de al menos ***d*.**



El puesto de control posee dos sensores **s1** y **s2 (**entradas del sistema), los cuales indican (con un 1) que hay una vela delante del sensor. Los sensores están dispuestos a la distancia ***d*** (ver figura)***.***

La cinta debe funcionar a velocidad alta mientras no hay una vela en el puesto de control. Cuando una vela es detectada por el sensor **s1**, la cinta debe comenzar a funcionar a velocidad baja hasta que la vela abandona el puesto. Para controlar la cinta se dispone de la salida ***cinta*** de dos bits (**c1** y **c0**), la que se codifica de la siguiente manera:

- 00 -> cinta detenida
- 10 -> cinta moviendose a velocidad baja
- 11 -> cinta moviendose a velocidad alta

Al llegar una vela al puesto de control, existen dos posibilidades: que la vela tenga el tamaño adecuado (mayor o igual a ***d***), o que sea más angosta, ante lo cual se deberá ensanchar el molde. Para realizar el ensanchado de las velas, la cinta se debe detener en el momento en que la vela se encuentra contenida entre ambos sensores (es decir, cuando ninguno de los sensores detecta la presencia de la vela) y se debe encender la salida ***requiere_ensanche*** (un 1 ensancha la vela), hasta que se detecte que la vela tiene diámetro mayor o igual a ***d*** (es decir, que ambos sensores detectan la presencia de la vela).

Una vez finalizado este proceso, la cinta debe continuar avanzando a velocidad baja hasta que la vela abandone el puesto de control. Como fue indicado anteriormente, una vez que la vela abandona el puesto de control, la cinta se debe mover a velocidad alta hasta que una nueva vela sea detectada por

el sensor **s1** y vuelva a comenzar el proceso**.**

Nota: Se puede asumir que la distancia entre las velas sobre la cinta es mucho mayor que *d.*

Se pide:
Diseñar y construir, utilizando la metodología del curso, un circuito que controle las salidas ***cinta*** *(**c1 c0**)* y ***requiere_ensanche*** a partir de las entradas de los sensores ***s1*** y ***s2***. Se dispone de flip flops tipo D y compuertas básicas.

Problema 2

Considere la siguiente estructura:

```
typedef struct {
    short valor;
    nodo* izq, der;
} nodo;
```

y la siguiente función:

```
short arboles_iguales(nodo* arbol1, nodo* arbol2) {
    if (arbol1 == NULL || arbol2 == NULL)
        return (arbol1 == arbol2);
    else if (arbol1->valor != arbol2->valor)
        return 0;
    else
        return (arboles_iguales(arbol1->izq, arbol2->izq) && arboles_iguales(arbol1->der, arbol2->der));
}
```

Parte a)

Compilar la función ***arboles_iguales*** en assembler 8086. Considere que los punteros son FAR, que los parámetros se reciben por stack y que el resultado se retorna en el stack. Es necesario preservar el valor de los registros.

El programa que va a invocar esta función lo hace con el siguiente pseudocódigo:
PUSH segmento arbol1
PUSH desplazamiento arbol1
PUSH segmento arbol2
PUSH desplazamiento arbol2
CALL arboles_iguales
POP resultado

Parte b)

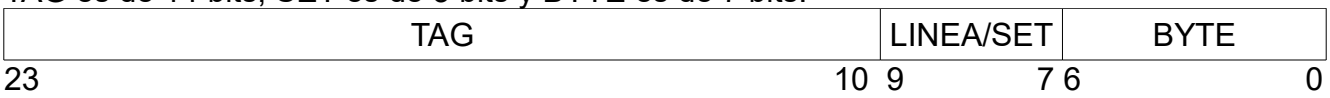
Indique el máximo consumo de stack de su solución para árboles de N y M nodos respectivamente.

Respuesta Pregunta 1:

El campo BYTE (palabra individual en una línea de caché o bloque de memoria) requiere 7 bits. Dado que las líneas son de 128B (128 = 2^7), se necesitan 7 bits para indicar el byte.
Dado que la memoria caché cuenta con 32 líneas y cuatro vías, se tienen 8 conjuntos. Se requieren por tanto 3 bits para indicar el SET.

Dado que la CPU opera con una capacidad de direccionamiento de 16 Mbytes, el campo TAG ocupa los restantes 14 bits.

TAG es de 14 bits, SET es de 3 bits y BYTE es de 7 bits.



Al ejecutar la instrucción `LOAD 0xA64CDE R1` se accede a memoria para cargar el primer dato.
La dirección `0xA64CDE` es `1010 0110 0100 1100 1101 1110 b`, los campos corresponden a TAG = `10100110010011 b` (0x2993), SET = `001 b` (0x1), BYTE = `1011110` b (0x5E).
La caché está inicialmente vacía, entonces no está en caché (miss). Se carga el bloque con tag 0x2993 en alguna línea del conjunto 0x1.

Al ejecutar la instrucción `LOAD 0xA64CFE R2` se accede a memoria para cargar el segundo dato.
La dirección `0xA64CFE` es `1010 0110 0100 1110 1111 1110` b, los campos corresponden a TAG = `10100110010011` b (0x2993), SET = `001` b (0x1), BYTE = `111 1110` b (0x7E).
Este bloque se cargó en el LOAD anterior (mismo conjunto y mismo tag). El dato buscado está en caché (hit).

La operación `MUL R1 R2` no accede a memoria.

Al ejecutar la instrucción `LOAD 0xA646FF R3` se accede a memoria para cargar el tercer dato.
La dirección `0xA646FF` es `1010 0110 0100 0110 1111 1110` b, los campos corresponden a TAG = `10100110010001` b (0x2991), SET = `101` b (0x5), BYTE = `111 1111` b (0x7F).
El conjunto 5 no tiene líneas cargadas, por lo tanto el dato buscado no está en caché (miss).

La operación `MUL R1 R3` no accede a memoria.

El hit-rate al ejecutar el código es ⅓.

Respuesta Pregunta 2:

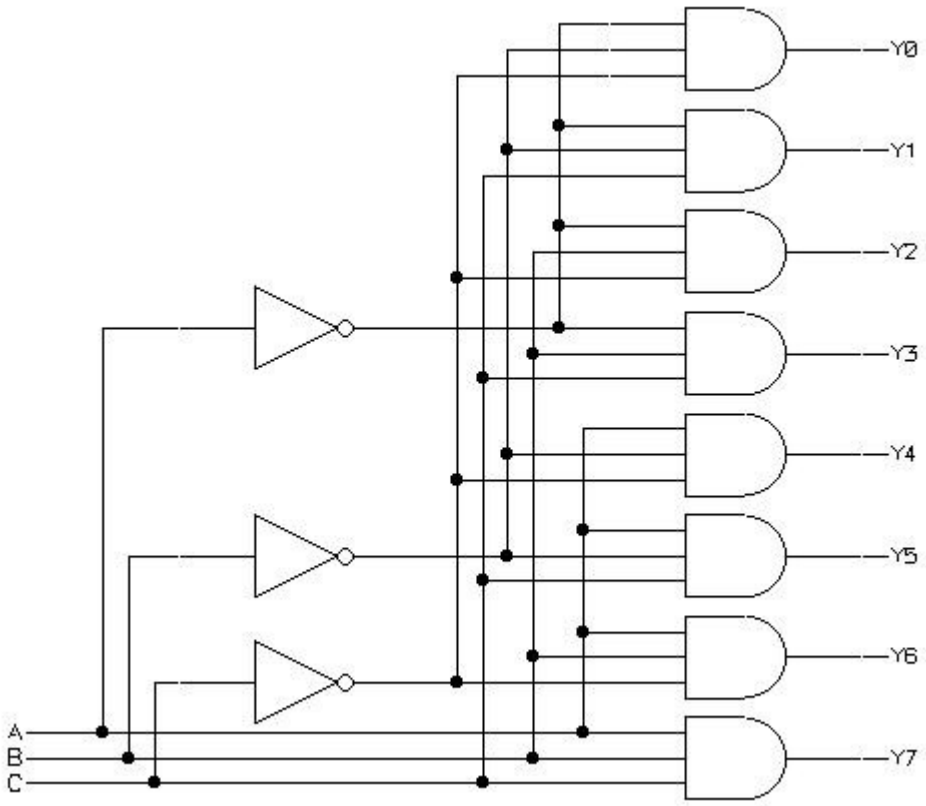
a) Dado que el formato de instrucción indica que se usan tres bits para los registros, el máximo número de registros que dispone la arquitectura es $2^3 = 8$ registros.

b) Cuando se utiliza direccionamiento directo, la dirección de memoria se incluye en la propia instrucción. Dado que el campo para especificar la dirección de memoria tiene 8 bits, se pueden direccionar hasta $2^8 = 256$ posiciones. El máximo espacio de direccionamiento es [0,255].

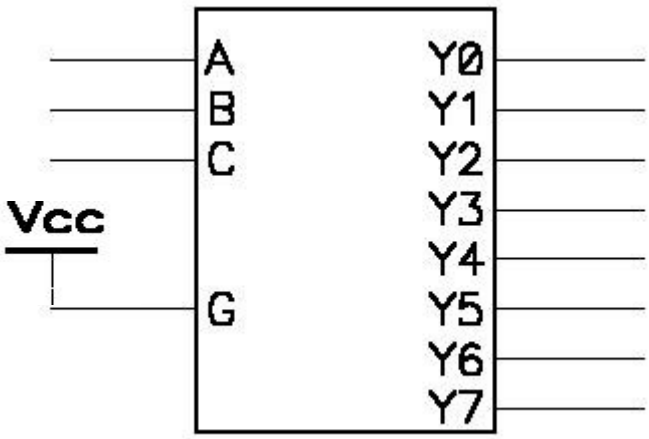
c) Cuando se utiliza direccionamiento indirecto por memoria, la dirección incluida en la instrucción contiene la dirección de memoria donde se almacena la dirección del dato. La memoria consta de palabras de 16 bits (por tratarse de una arquitectura de 16 bits), por lo cual se pueden almacenar direcciones de hasta $2^{16} = 65536$ posiciones. El máximo espacio de direccionamiento es [0,65535].

Respuesta Pregunta 3:

a)



b)



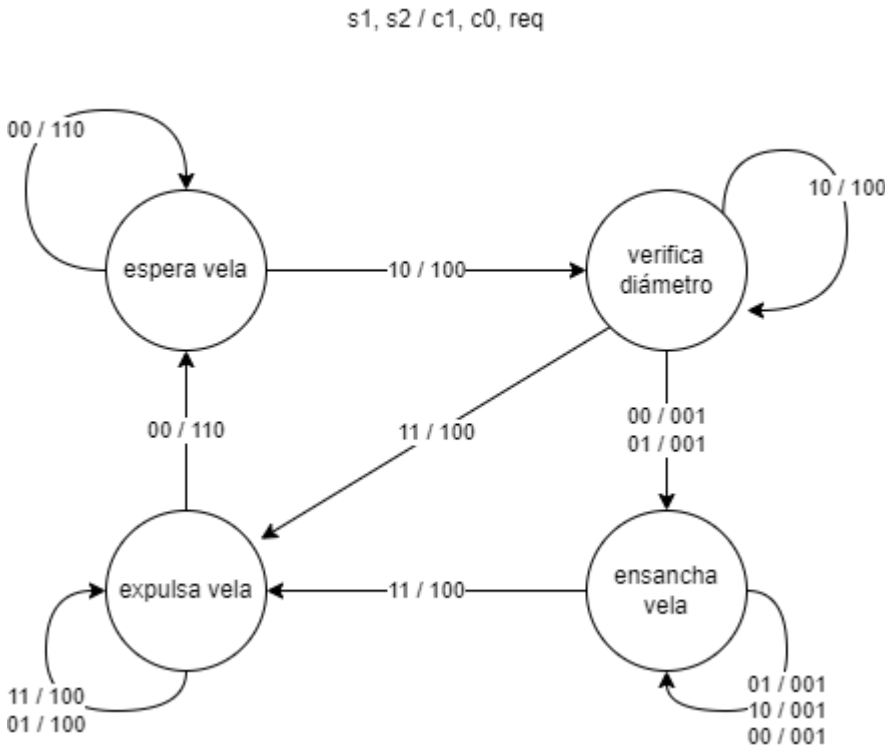
Respuesta Pregunta 4:

```
CRTL EQU ...
DATA EQU ...
MASK_CRTL EQU 8

xor BX, BX
mov ES, BX
cli
loop:
     cmp BX, 1024
     jae fin
     mov DX, CRTL
polling_ctrl:
     in AX, DX
     and AX, MASK_CRTL
     jz polling_ctr
     mov DX, DATA
     in AX, DX
     mov ES:[BX], AX
     add BX, 2
     jmp loop
fin:
sti
…
```

Solución Problema 1:

Usaremos la notación s1 s2 / c1 c0 requiere_ensanche para indicar las entradas y salidas en las transiciones de la máquina de estados.



Nota 1: la transición 01 / 001 entre el estado “verifica diámetro” y “ensancha vela” puede ser considerada también como “imposible” y no ser considerada en el diagrama y la tabla.

Nota 2: el estado inicial es “espera vela”.

Tabla de estados:

Estado actual	Entrada (S1, S2)		Próximo estado	Salida (C1, C0, req)		
Espera vela	0	0	Espera vela	1	1	0
Espera vela	1	0	Verifica diámetro	1	0	0
Verifica diámetro	0	0	Ensancha vela	0	0	1
Verifica diámetro	0	1	Ensancha vela	0	0	1
Verifica diámetro	1	0	Verifica diámetro	1	0	0
Verifica diámetro	1	1	Expulsa vela	1	0	0
Expulsa vela	0	0	Espera vela	1	1	0
Expulsa vela	0	1	Expulsa vela	1	0	0
Expulsa vela	1	1	Expulsa vela	1	0	0
Ensancha vela	0	0	Ensancha vela	0	0	1

Ensancha vela	0	1	Ensancha vela	0	0	1
Ensancha vela	1	0	Ensancha vela	0	0	1
Ensancha vela	1	1	Expulsa vela	1	0	0

Como la solución tiene 4 estados, se precisan 2 bits para codificarlos, por lo tanto se precisan 2 flip flops.

La codificación de estados es la siguiente:

00 -> Espera vela
01 -> Verifica diámetro
10 -> Expulsa vela
11 -> Ensancha vela

La tabla de verdad para el problema queda:

Estado actual (Q1, Q0)		**Entrada (S1, S2)**		**Próximo estado (D1, D0)**		**Salida (C1, C0, req)**		
0	0	0	0	0	0	1	1	0
0	0	0	1	X	X	X	X	X
0	0	1	0	0	1	1	0	0
0	0	1	1	X	X	X	X	X
0	1	0	0	1	1	0	0	1
0	1	0	1	1	1	0	0	1
0	1	1	0	0	1	1	0	0
0	1	1	1	1	0	1	0	0
1	0	0	0	0	0	1	1	0
1	0	0	1	1	0	1	0	0
1	0	1	0	X	X	X	X	X
1	0	1	1	1	0	1	0	0
1	1	0	0	1	1	0	0	1
1	1	0	1	1	1	0	0	1
1	1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0	0

Mapas de Karnaugh:

q1q0\ s1s2	00	01	11	10
00	0	X	X	0
01	1	1	1	0
11	1	1	1	1
10	0	1	1	X

D1 = s2+!s1q0+q1q0

q1q0\ s1s2	00	01	11	10
00	0	X	X	1
01	1	1	0	1
11	1	1	0	1
10	0	0	0	X

D0 = q0!s1+s1!s2

q1q0\ s1s2	00	01	11	10
00	1	X	X	1
01	0	0	X	1
11	0	0	1	0
10	1	1	1	X

C1 = !q0+s1s2+!q1s1

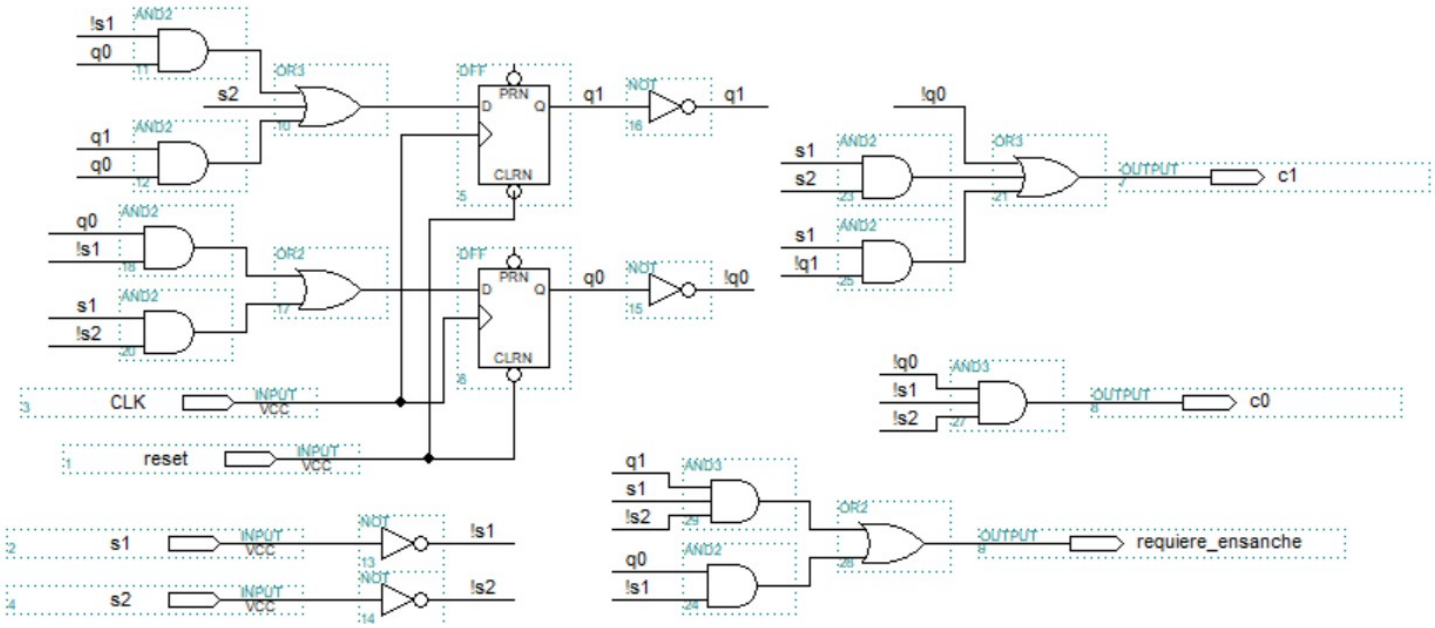
q1q0\ s1s2	00	01	11	10
00	1	X	X	0
01	0	0	0	0
11	0	0	0	0
10	1	0	0	X

C0 = !q0!s1!s2

q1q0\ s1s2	00	01	11	10
00	0	X	X	0
01	1	1	0	0
11	1	1	0	1
10	0	0	0	X

Requiere_ensanche = q0!s1+ q1s1!s2

Circuito:



Solución Problema 2:

Parte a)

```
arboles_iguales proc
  push BP
  mov BP, SP
  push AX
  push BX
  push CX
  push SI
  push DS
  push ES
  mov BX, [BP+4]  ; BX=desp arbol2
  mov AX, [BP+6]  ; AX=segmento arbol2
  mov SI, [BP+8]  ; SI=desp arbol1
  mov CX, [BP+10] ; CX=segmento arbol1
  mov DS, CX      ; DS=seg a1
  mov ES, AX      ; ES=seg a2
  or CX, SI
  jz esnull
  or AX, BX
  jnz else       ; if (arbol1 == NULL || arbol2 == NULL)
esnull:
  cmp BX, SI
  jne ret0
  mov CX, [BP+10]
  cmp CX, [BP+6]
  jne ret0
  mov AX, 1
  jmp fin
else:
  mov AX, [SI]
  cmp AX, ES:[BX]
  jne ret0       ; if (arbol1->valor != arbol2->valor)
  push [SI+4]
  push [SI+2]
  push ES:[BX+4]
  push ES:[BX+2]
  call arboles_iguales
  pop AX         ; AX=arboles_iguales(arbol1->izq, arbol2->izq)
  cmp AX, 0
  je ret0
  push [SI+8]
  push [SI+6]
  push ES:[BX+8]
  push ES:[BX+6]
  call arboles_iguales
  pop AX         ; AX=arboles_iguales(arbol1->der, arbol2->der)
  jmp fin
```

```
ret0:
  xor AX, AX
fin:
  mov [BP+10], AX  ; resultado
  mov AX, [BP+2]
  mov [BP+8], AX   ; dir ret
  pop ES
  pop DS
  pop SI
  pop CX
  pop BX
  pop AX
  pop BP
  add SP, 6
  ret
arboles_iguales endp
```

Parte B)

Tanto el paso base como los pasos recursivos (rama izquierda y rama derecha) consumen 4 palabras por parámetros, 1 palabra de dirección de retorno y 7 palabras de contexto.

En total son 12 palabras o 24 bytes por invocación.

Si ambos árboles tienen la misma cantidad de nodos (N=M) el peor caso se da cuando ambos árboles degeneran en una lista (un solo hijo por nodo) y son iguales.

En ese caso el consumo es de N pasos recursivos más el paso baso, o sea (N+1)*24 bytes

En el caso que N<>M tomo sin perder generalidad N<M. El peor caso se da cuando el arbol de N nodos degenera en lista, y el árbol de M nodos incluye al otro árbol (los datos coinciden) y en todos los nodos salvo la hoja final si el árbol de N nodos tiene hijo por derecha el árbol de M nodos no tiene hijo por izquierda.

En este caso nuevamente se consumen (N+1) pasos del algoritmo para determinar que no son iguales, o sea (N+1)*24 bytes