Examen de Arquitectura de Computadoras

22 de diciembre de 2021

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.**
- **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 procesador con pipeline. Explique la técnica de predicción de saltos e indique qué tipo de hazard mitiga. Describa **una** de las siguientes técnicas de predicción de saltos:

- Predicción basada en la condición del salto
- Conmutar taken/not taken
- Branch History Table

Pregunta 2

Decodifique el siguiente mensaje codificado con Hamming de 4 bits de datos por paquete, indicando para cada paquete el dato recibido y la corrección que deba realizarse si corresponde.

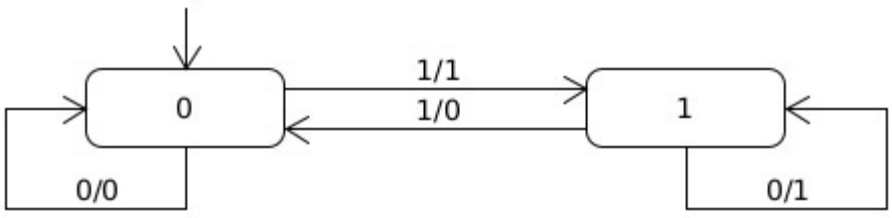
* 0x12
* 0x3D
* 0x52

Pregunta 3

Describa las memorias SRAM y DRAM. Señale al menos tres diferencias entre ambas.

Pregunta 4

Utilizando Flip Flop tipo JK y siguiendo la metodología del curso, implemente la siguiente máquina de estados:



Problema 1

Se desea controlar el techo corredizo de un invernadero a través de un sistema de control basado en un procesador dedicado.

El techo se utiliza para evitar la entrada excesiva de agua durante los días de lluvia, sin embargo, cierta cantidad de lluvia es tolerada pues se sabe que no dañará los cultivos. Para calcular si se debe cerrar el techo, se asume que la tierra absorbe **5 mm** de lluvia por minuto. Si llueve a mayor intensidad, el agua se acumula en la superficie de la tierra. El techo se debe cerrar en caso de que la lluvia acumulada sobre la tierra sea mayor a **1 cm**. Adicionalmente, se debe disparar una alarma si el agua acumulada supera los **2 cm**. La alarma se debe mantener encendida hasta que el techo se cierre completamente.

Para medir la lluvia se dispone de un pluviómetro digital. Este pluviómetro contiene un pequeño recipiente cilíndrico para almacenar la lluvia. Cuando se supera **1 mm** de lluvia recibida, el agua se drena y se invoca la interrupción ***pluviometro()*.**

El techo se cierra o se abre con un motor que es controlable a través del puerto de entrada/salida de solo escritura, de 8 bits, en la dirección **CONTROL**. El bit 1 controla el encendido del motor (un 1 activa el motor), mientras que el bit 0 controla el sentido del movimiento (un 1 abre el techo, un 0 lo cierra). Adicionalmente, el bit 4 del puerto **CONTROL** permite manejar la alarma (un 1 enciende la alarma, un 0 la apaga). Cuando el techo se abre o se cierra completamente se invoca a la rutina de interrupción ***tope().***

Cuando el motor está en funcionamiento se debe monitorear su temperatura para minimizar los riesgos de desperfectos por sobrecalentamiento. La temperatura del motor se puede leer en el puerto de entrada/salida de solo lectura, de 16 bits, en la dirección **TEMP_MOTOR**. El sensor de temperatura indica en el bit menos significativo si hay una lectura válida o no (el bit en 1 significa que hay). Los siguientes 10 bits indican la temperatura en décimas de grado celsius (como entero sin signo). Los 5 bits más significativos son reservados y se deben descartar. Si la temperatura es mayor a 80 grados celsius, se debe aplicar refrigeración al motor, lo cual se logra encendiendo el bit 7 del puerto **CONTROL**.

El techo se debe abrir nuevamente una vez que pasaron 10 minutos sin lluvia.

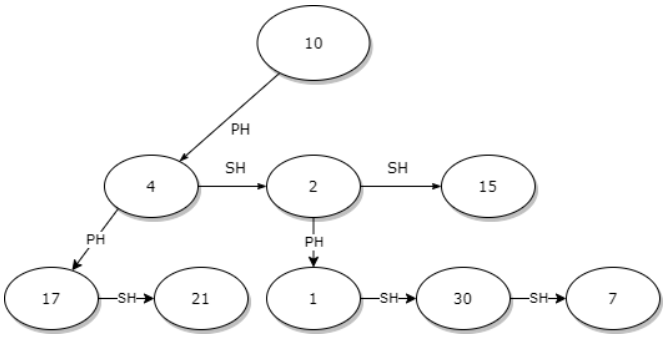
Se dispone de un reloj externo de 5 Hz que interrumpe al procesador invocando a la rutina ***tiempo()***.

Se pide:

Diseñar una solución a la realidad planteada y escribir en un lenguaje de alto nivel (preferentemente C) todas las rutinas necesarias.

Problema 2

Se dispone de un procedimiento en C que calcula la cantidad de elementos por nivel de un árbol general n-ario, representado mediante la semántica primer hijo-siguiente hermano (PH-SH). Como resultado de la aplicación del procedimiento que realiza el conteo por niveles se actualiza un arreglo de enteros, cuyo tamaño es la cantidad de niveles del árbol. Adicionalmente, el procedimiento retorna la cantidad de elementos procesados (los elementos del árbol que se le pasa como parámetro). La Figura 1 presenta un ejemplo de árbol general representado con la semántica PH-SH.



A continuación se presenta la estructura utilizada para representar el árbol, y el código en alto nivel del procedimiento que realiza el conteo por niveles, el cual utiliza una función auxiliar recursiva.

```
Struct nodo{
   int dato;
   nodo* PH;
   nodo* SH;
}
typedef arbol* nodo;

int elementos_por_nivel(arbol a, int cantidad_niveles, int* cantidad_por_nivel){
   int cantidad_nodos, i;

   elementos_por_nivel_aux(a, cantidad_niveles, cantidad_por_nivel, 0);
   cantidad_nodos = 0;
   for (i = 0; i < cantidad_niveles; i++)
      cantidad_nodos += cantidad_por_nivel[i];
   return cantidad_nodos;
}

void elementos_por_nivel_aux(arbol* a, int cantidad_niveles, int* cantidad_por_nivel, int
nivel_actual){
   if(a->PH != 0) // Primero se recorre en profundidad
      elementos_por_nivel_aux(a->PH, cantidad_niveles, cantidad_por_nivel, nivel_actual+1);
   if(a->SH != 0) // Luego se recorren los hermanos
      elementos_por_nivel_aux(a->SH, cantidad_niveles, cantidad_por_nivel, nivel_actual);
   cantidad_por_nivel[nivel_actual] += 1;
}
```

Se pide:

a) Compilar el código presentado a Assembler 8086.
b) Calcular el consumo máximo de stack al realizar la invocación de elementos_por_nivel, utilizando como dato (solamente para esta parte del problema) que la cantidad máxima de hijos directos de un nodo es K.

Aclaraciones:

- Los elementos de cantidad_por_nivel son consecutivos y relativos al segmento DS.
- Las llamadas a los procedimientos y funciones son todas near.
- Los parámetros de los procedimientos se colocan en el stack en el orden en que aparecen en el encabezado.

Solución

Respuesta Pregunta 1

La técnica de predicción de saltos consiste en intentar determinar, en base a un cierto algoritmo o criterio, si el salto se va a tomar o no. El objetivo es aumentar la probabilidad de cargar las instrucciones correctas al ejecutar una bifurcación. Esta técnica mitiga el impacto de los hazards de control.

La técnica de **predicción basada en la condición del salto** consiste en, una vez decodificada la instrucción, determinar si el salto se tomará o no según qué tipo de condición se está evaluando para efectuar el salto (por ej, en 8086: JE, JNE, JG, JLE, etc). Esta técnica de predicción es efectiva ya que se ha constatado mediante ejecución de *benchmarks,* que ciertas condiciones de salto son más propensas a producir efectivamente el salto que otras.

La técnica de **conmutar taken/not taken** utiliza la historia reciente de los saltos, mediante una máquina de cuatro estados que “recuerda” si el último salto fue tomado o no. La predicción debe fallar dos veces seguida para cambiarla.

La técnica de **Branch History Table** es un mecanismo anterior en el que se almacena en una tabla la información de un cierto conjunto de saltos recientemente evaluados. Aplica el algoritmo “taken/no taken” sobre el comportamiento de cada salto individual, es decir según la historia de cada salto y no la historia de todos juntos, permitiendo que la decisión sobre un salto particular no sea interferida por las decisiones tomadas sobre otros saltos.

(nota: en el examen alcanzaba con poner una de las descripciones).

Respuesta Pregunta 2

Considerando el ordenamiento de bits que forman cada paquete de Hamming de 4 bits a4a3a2p3a1p2p1 podemos calcular el sindromes de cada paquete (x es xor):

s2=p3xa4xa3xa2
s1=p2xa4xa3xa1
s0=p1xa4xa2xa1

* 0x12 = 0010010 paquete: 0010

s2=0x0x0x1 = 1
s1=1x0x0x0 = 1
s0=0x0x1x0 = 1

Por lo tanto el bit 7 está mal y se corrige a 1010010 = 0x52 => paquete 1010

* 0x3D = 0111101 paquete: 0111

s2=1x0x1x1 = 1
s1=0x0x1x1 = 0
s0=1x0x1x1 = 1

Por lo tanto el bit 5 está mal y se corrige a 0101101 = 0x2D => paquete 0101

* 0x52 = 1010010 paquete 1010.

No tiene error (el sindrome dará 0 pues corresponde al primer paquete ya corregido)

Respuesta Pregunta 3

Una memoria SRAM, por Static Random Access Memory, es una memoria contruída en base a un arreglo de flip-flops tipo D. Típicamente poseen m entradas de dirección y n entradas/salidas de datos, para un total de 2^m palabras de n bits.

Una memoria DRAM por Dynamic RAM, es una memoria construída en base a un artilugio electrónico consistente en aprovechar la capacidad parásita de los transistores para almacenar una carga eléctrica que representa el valor del bit. Los bits se organizan en una matriz, por lo que para acceder a un bit es necesario proporcionar la fila y la columna, lo que se hace de a uno por vez utilizando las entradas de direccion y las de control CAS y RAS.
Algunas de las diferencias son:
- Las SRAM son más rápidas que las DRAM.
- Las SRAM se organizan como un arreglo de palabras de n bits, mientras que las DRAM como una matriz de bits individuales.
- Las SRAM mantienen el valor del bit mientras tengan energía eléctrica, mientras que eso no alcanza para las DRAM que además necesitan de ser leídas periodicamente para reconstruir el valor de los bits almacenados (refresh).
- Las DRAM son más densas que las SRAM (aprox: 1 transistor por bit vs 6 transitores por bit).
- Las SRAM son mas caras que las DRAM.

Respuesta Pregunta 4

Estado (t)	Entrada	Estado (t+1)	Salida
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Q (t)	Entrada	Q (t+1)	J	K	Salida
0	0	0	0	x	0
0	1	1	1	x	1
1	0	1	x	0	1
1	1	0	x	1	0

Q \ E	0	1
0	0	1
1	X	X

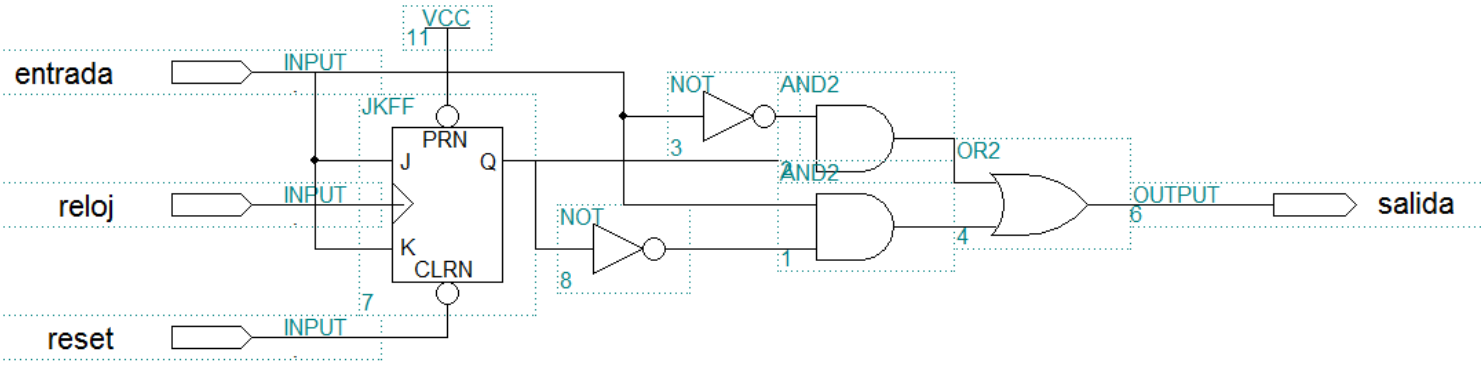
J = E

Q \ E	0	1
0	X	X
1	0	1

K = E

Q \ E	0	1
0	0	1
1	1	0

Salida = Q’E+QE ’



Solución Problema 1

```
#define ABIERTO 0
#define CERRANDO 1
#define CERRADO 2
#define ABRIENDO 3
#define UN_CM 10
#define DOS_CM 20
#define DIEZ_MINUTOS 3000 // 5 * 60 * 10
#define DOCE_SEGUNDOS 60
#define MAX_TEMP_DG 800   // 80 Grados en décimas de grado

int ticsSinLluvia = 0;
char ticsMMTierra = 0;
char estado = ABIERTO;
short mmAgua = 0;
bool llegoTope = false;
char hayAlarma = 0;
char hayRefrigeracion = 0;

void main(){
  OUT(CONTROL, 0); // inicializa estado del hardware de E/S
  // instalar interrupciones
  enable();
  while (true){
      switch (estado){
            case ABIERTO:
                  if (mmAgua > UN_CM)
                        estado = CERRANDO;
                  break;
            case CERRANDO:
                  if (llegoTope){
                        estado = CERRADO;
                        llegoTope = false;
                        hayAlarma = 0;
                        OUT(CONTROL, 0);
                        break;
                  }
                  if (mmAgua > DOS_CM)
                        hayAlarma = 0x10;
                  calcularRefrigeracion();
                  OUT(CONTROL, hayRefrigeracion | hayAlarma | 0x2);
                  break;
            case CERRADO:
                  if (ticsSinLluvia >= DIEZ_MINUTOS)
                        estado = ABRIENDO;
                  break;
            case ABRIENDO:
                  if (llegoTope){
                        estado = CERRADO;
                        llegoTope = false;
                        OUT(CONTROL, 0);
                        break;
                  }
                  calcularRefrigeracion();
                  OUT(CONTROL, hayRefrigeracion | 0x03); // abrir techo
                  break;
      }
```

```
  }
}

void calcularRefrigeracion(){
  short temp = IN(TEMP_MOTOR);
  if (temp & 1){ // si lectura es válida
      short tempDecGrados = ((temp >> 1) & 0x3FF);
      if (tempDecGrados > MAX_TEMP_DG){
            hayRefrigeracion = 0x40;
      }
      else {
            hayRefrigeracion = 0;
      }
  }
}

interrupt void tiempo(){
  ticsMMTierra++;
  if (ticsMMTierra >= DOCE_SEGUNDOS){
      // la tierra drena 5mm/minuto -> 1mm cada 12 segundos
      ticsMMTierra = 0;
      if (mmAgua > 0)
            mmAgua--;
  }
  ticsSinLluvia++;
}

interrupt void pluviometro(){
  mmAgua++;
  ticsSinLluvia = 0;
}

interrupt void tope(){
  llegoTope = true;
}
```

Nota: esta solución no controla que pueda empezar a llover mucho mientras se está abriendo el techo y se de la condición para volver a cerrarlo. En caso de querer realizar ese control se debe agregar la verificación (y cambio de estado si se da) que se hace en el caso de estado = ABIERTO al final del código del caso estado = ABRIENDO.

Solución Problema 2

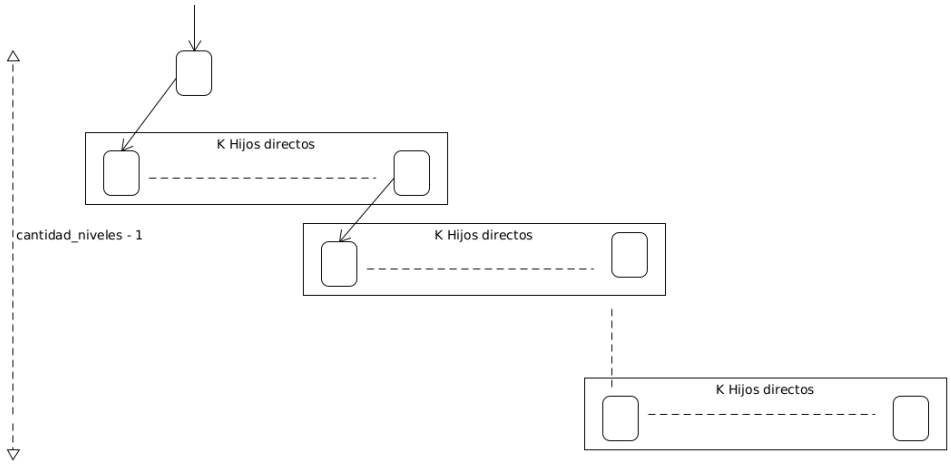
Parte a)

```
elementos_por_nivel PROC
   PUSH BP
   MOV BP, SP
   PUSH AX
   PUSH BX
   PUSH CX
   PUSH SI

   XOR CX, CX          //cantidad_nodos
   MOV BX, [BP+4]      //cantidad_por_nivel
   MOV AX, [BP+6]      //cantidad_niveles
   PUSH [BP+8]         //a
   PUSH AX             //cantidad_niveles
   PUSH BX             //cantidad_por_nivel
   PUSH CX             // 0
   CALL elementos_por_nivel_aux
   SHL AX,1            // cantidad de niveles * 2 para la comparación del bucle
   XOR SI, SI          //i=0
bucle:
   CMP SI, AX
   JGE finbucle
   ADD CX, [BX+SI]     //cantidad_nodos += cantidad_por_nivel[i]
   ADD SI, 2
   JMP bucle
finbucle:
   MOV AX, [BP+2]
   MOV [BP+6], AX      // acomodo dirección de retorno
   MOV [BP+8], CX      // guardo cantidad_nodos en parámetro de retorno
   POP SI
   POP CX
   POP BX
   POP AX
   POP BP
   ADD SP, 4
   RET
elementos_por_nivel ENDPROC

elementos_por_nivel_aux PROC
   PUSH BP
   MOV BP, SP
   PUSH AX
   PUSH BX
   PUSH SI
   PUSH DI

   MOV SI, [BP+10]          //a
   MOV AX, [BP+8]           //cantidad_niveles
   MOV DI, [BP+6]           //cantidad_por_nivel
   MOV BX, [BP+4]           //nivel_actual
   CMP WORD PTR [SI+2], 0
   JE sin_hijos
```

```
   PUSH [SI+2] //a->PH
   PUSH AX
   PUSH DI
   INC BX
   PUSH BX
   CALL elementos_por_nivel_aux
   DEC BX
sin_hijos:
   CMP WORD PTR [SI+4], 0
   JE sin_hermanos
   PUSH [SI+4]             //a->SH
   PUSH AX
   PUSH DI
   PUSH BX
   CALL elementos_por_nivel_aux
sin_hermanos:
   SHL BX, 1
   INC WORD_PTR [DI+BX]    //cantidad_por_nivel[nivel_actual] += 1
   MOV AX, [BP+2]
   MOV [BP+10], AX         //acomodo dirección de retorno
   POP DI
   POP SI
   POP BX
   POP AX
   POP BP
   ADD SP, 8
   RET
elementos_por_nivel_aux ENDPROC
```

Parte b)

Suponemos que el arbol está bien formado y por tanto el nodo raíz no tiene hermanos.
El peor caso se da cuando para cada primer nodo se recorren K-1 siguientes hermanos y en el último hermano se pasa a su primer hijo y se repite esta secuencia hasta completar la cantidad de niveles., según se muestra en el diagrama. Como el primer nodo no tiene hermanos, la cantidad de veces que se repite esto es *cantidad_niveles* – 1.



Una llamada recursiva de *elementos_por_nivel_aux* para PH, así como también para SH tiene 4 parámetros, entonces teniendo en cuenta la dirección de retorno el consumo de una llamada es 5*2bytes=10bytes.

Luego, al comienzo del procedimiento recursivo se guarda el contexto de 5 registros en el stack que serían otros 10bytes. Em consecuencia cada llamada recursiva, sea por PH o por SH consume 20 bytes.

llamada a nodo raiz + (llamada al PH del nodo + (K – 1) llamadas a sus hermanos) * (*cantidad_niveles* – 1)
(1 * 20 + (1 * 20 + (K – 1) * 20) * (*cantidad_niveles* – 1)) bytes =
(1 + K * (*cantidad_niveles* – 1)) * 20 bytes =

(notar que no se realizan llamadas de paso base a los punteros NULL).

Finalmente, se debe agregar el consumo de stack en la ejecución de *elementos_por_nivel*, el cual consiste en 10 bytes de contexto, 6 bytes de parámetros y 2 bytes del IP, es decir, 18 bytes en total.
Entonces, el consumo máximo de stack para una llamada al procedimiento *elementos_por_nivel* en las condiciones planteadas es:

18 bytes + (K * (*cantidad_niveles* - 1)) * 20 bytes.

Solución Alternativa Problema 2

Parte a)

```
elementos_por_nivel PROC
  POP AX ; AX=dir retorno
  XOR CX, CX
  PUSH CX
  CALL elementos_por_nivel_aux   ;se asume conserva AX
  SHL CX,1                       ; CX = cant niveles*2
  XOR DX, DX                     ; DX:cantidad_nodos=0
  XOR BX, BX                     ; i=0
for:
  CMP BX, CX
  JGE finfor
  ADD DX, [BX+DI]                ; cantidad_nodos += cantidad_por_nivel[i]
  ADD BX, 2
  JMP for
finfor:
  PUSH DX
  PUSH AX
  RET
elementos_por_nivel ENDPROC

; deja en BP=niv_actual, DI=cant_por_nivel, CX=cant_niveles
elementos_por_nivel_aux PROC
  POP DX                         ; dir_ret
  POP BP                         ; niv_actual
  POP DI                         ; cant_por_nivel
  POP CX                         ; cant_niveles
  POP SI                         ; a
  PUSH DX                        ; dejo dir_ret
  CMP WORD PTR [SI+2], 0
  JE fin_if1
  PUSH SI ; guardo contexto (a)
  PUSH DS:[SI+2]
  PUSH CX
  PUSH DI
  INC BP
  PUSH BP
  CALL elementos_por_nivel_aux
  DEC BP
  POP SI                         ; recupero contexto (a)
fin_if1:
  CMP WORD PTR [SI+4], 0
  JE fin_if2
  PUSH DS:[SI+4]
  PUSH CX
  PUSH DI
  PUSH BP
  CALL elementos_por_nivel_aux
fin_if2:
  MOV BX, DI
  INC WORD PTR [BX+DI]           ; cantidad_por_nivel[nivel_actual] += 1;
```

```
  RET
elementos_por_nivel_aux ENDPROC
```

parte b)

Por las mismas razones expuestas en la anterior solución, resulta que el consumo de stack de la función recursiva auxiliar es:

llamada a nodo raiz + (llamada al PH del nodo + (K – 1) llamadas a sus hermanos) * (*cantidad_niveles* – 1)

a lo que se le debe sumar el consumo de la rutina inicial,

En este caso la rutina *elementos_por_nivel* se tienen 6 bytes de parametros y 2 de la dirección de retorno, pero estos se recuperan del stack (la direccion queda en AX y los parametros se usan para la primer llamada a elementos_por_nivel_aux). Por tanto si el consumo del resto es mayor a 8 bytes esta invocación no incide en el total.

Para la rutina elementos_por_nivel_aux se tienen 8 bytes de parámetros y 2 de la dirección de retorno, pero los parámetros se quitan del stack y solo se deja la dirección de retorno.
Para el paso recursivo en profundidad se guarda como contexto adicional el puntero al arbol.
Para el paso recursivo por hermanos no se guarda contexto adicional.

Por lo tanto el consumo maximo para la función recursiva auxiliar será:

2 + (4 + 2 * (K – 1)) * (*cantidad_niveles* – 1) =
2 + 2 * (K + 1) * (*cantidad_niveles* – 1)

y en consecuencia el consumo total, incluyendo la función *elementos_por_nivel* es el mismo (por lo ya dicho):
2 bytes + 2 * (K + 1) * (*cantidad_niveles* – 1) bytes