ARQUITECTURA DE COMPUTADORAS

LABORATORIO

Arquitectura 8086

Autor

Nombre	CI
Martina Bernini Quesada	5.041.130-6

Índice

1. Introducción

1.1. Propósito

La presente tarea de laboratorio esta enfocada en la arquitectura 8086 y su lenguaje ensamblador. El laboratorio aborda la compilación manual desde el lenguaje C, la interacción con E/S y la implementación de rutinas recursivas.

El informe se centra en el desarrollo de un programa llamado "Manejador y Ordenador de Árboles de Búsqueda"(MOAB), que administra un Árbol Binario de Búsqueda (ABB). El programa realiza diversas operaciones, como agregar números al árbol y gestionar formatos de almacenamiento interno. También mantiene una bitácora de ejecución.

1.2. Preguntas teoricas

1. Explique cuáles son las consecuencias de cambiar la estructura 'nodo' en el modo dinámico y guardar los índices de los hijos izquierdo/derecho en un único byte.

   Hay dos consecuencias sustanciales.
   1) **Mayor capacidad de almacenamiento**. Al ocupar cada nodo unicamente 4 bytes (2 bytes para almacenar el valor y un byte para cada hijo), se incrementa la cantidad de nodos que se pueden almacenar dentro de AREA_DE_MEMORIA. Pasando los mismos a ser 2048/2 en la realidad de este laboratorio.
   2) **Menor capacidad de direccionamiento**. Por otro lado, al guardar los indices de los hijos derecho/izquierdo en un unico byte, solo podria referenciarse hasta el nodo 255. Si se agregan mas nodos, los mismos quedarian sin poder referenciarse.
   3) **Registros**. Cuando se emplea un byte para almacenar los índices, ya no es posible utilizar los registros AX, BX, CX y DX, ya que estos registros almacenan 2 bytes de memoria. En su lugar, se deben utilizar los registros AL/AH, BL/BH y así sucesivamente para trabajar con datos de 1 byte.

2. Explique qué cambios se deben realizar en su programa para que el árbol maneje enteros positivos de 16 bits (en lugar de enteros en complemento a dos).

   Los cambios que deben llevarse a cabo son los siguientes:
   1) **Cambio en las instrucciones de salto después de las comparaciones.** En la estructura original, se utilizaron instrucciones como JL/JG, que operan con enteros con signo. Para adaptarse a nuestra nueva situación, estas instrucciones deben reemplazarse por JB/JA, que son sus equivalentes pero operan con enteros sin signo.
   2**) Rango de datos**. Al cambiar a enteros sin signo, es fundamental asegurarse de que los valores ingresados sean válidos y se encuentren dentro del rango de datos representables. A su vez, seria necesario cambiar el valor con el cual se inicializa el arbol, ya que pasaria a ser un valor posible para ingresar.
   3) **Manejo de errores**. Debería implementarse un mecanismo de manejo de errores que detecte y gestione la inserción de números negativos en el árbol, evitando así situaciones no deseadas.

3. ¿Cuál es la máxima altura que soporta el programa en modo estático (en función de AREA_DE_MEMORIA)?

   La altura máxima que el programa puede soportar en modo estático se puede calcular de la siguiente manera, como se desarrolló en la tarea: dado un nodo en la posición 'i', se puede acceder a su rama izquierda y derecha utilizando las siguientes fórmulas: $2 \cdot i + 2$ y $2 \cdot i + 4$ respectivamente.

   Un árbol con altura máxima se puede generar cuando los nodos se organizan de manera que formen una cadena lineal hacia la izquierda. Esto se debe a que este recorrido avanza más "*lento*" en la memoria. La secuencia de espacio de almacenamiento para cada nodo sigue la siguiente sucesión:

- Nodo base: $P_1 = 0$
- Siguiente nodo: $P_n = 2 \cdot P_{n-1} + 2$

Avanzando un par de pasos en la sucesión, obtenemos los siguientes resultados:
$P_2 = 2 \cdot P_1 + 2 = 2$
$P_3 = 2 \cdot P_2 + 2 = 2 \cdot 2 + 2 = 2^2 + 2$
$P_4 = 2 \cdot P_3 + 2 = 2 \cdot (2^2 + 2) + 2 = 2^3 + 2^2 + 2 = 2^4 - 2$
$P_5 = 2 \cdot P_4 + 2 = 2 \cdot (2^3 + 2^2 + 2) + 2 = 2^4 + 2^3 + 2^2 + 2 = 2^5 - 2$

Luego, podemos deducir que $P_n = 2^n - 2$.
Por lo tanto,

$$2^n - 2 \leqslant 2 \cdot AREA\_DE\_MEMORIA$$
$$2^n \leqslant 2 \cdot AREA\_DE\_MEMORIA$$
$$log_2(2^n) \leqslant log_2(2 \cdot AREA\_DE\_MEMORIA)$$
$$n \leqslant 2 \cdot log_2(AREA\_DE\_MEMORIA) + log_2(2) = log_2(AREA\_DE\_MEMORIA) + 1$$

Finalmente, podemos agregar 'n' nodos que cumplan con la ecuación anterior, generando una altura máxima de $log_2(AREA\_DE\_MEMORIA) + 1$

4. ¿Cuál es la máxima altura que soporta el programa en modo dinámico (en funcion de AREA_DE_MEMORIA)?

   En este modo, los nodos se añaden uno tras otro en el espacio de memoria sin utilizar la sucesión mencionada en el punto anterior. Por lo tanto, un árbol de altura máxima se puede generar agregando tantos nodos como permita 'AREA_DE_MEMORIA'.

   Profundicemos en el cálculo: cada nodo consume 6 bytes de memoria en el arreglo. Por lo tanto, en total se pueden agregar 'floor(AREA_DE_MEMORIA / 3)' nodos, lo que representa la altura máxima del árbol.

5. ¿Cuál es el máximo consumo de stack al invocar la rutina que imprime un árbol en modo estático de altura M?

   El máximo consumo de memoria en el stack al llamar a la función 'imprimirArbol' se alcanza cuando el árbol está desequilibrado al extremo, es decir, cuando todos los nodos forman una línea recta hacia la izquierda o la derecha, con una altura de M nodos.

   Para fines de esta explicación, tomaré el caso de impresión de menor a mayor y supondremos que el árbol está completamente desequilibrado hacia la izquierda. Los otros casos son análogos.

   El consumo de memoria en el stack se desglosa de la siguiente manera:
   1) Llamada inicial al procedimiento imprimirArbol
   2) Si el nodo no es nulo
   2.1) $\longrightarrow$ Realizo un push con el valor del nodo.
   2.2) $\longrightarrow$ Llamo de manera recursiva a la función para la rama izquierda.

   Es evidente que el paso 2 se repite para cada uno de los M nodos sucesivos, lo que resulta en un consumo máximo total de memoria en el stack de $(1 + 2 \cdot M) \cdot 2$ bytes.

6. ¿Cuál es el máximo consumo de stack al invocar la rutina que imprime un árbol en modo dinámico de altura M?

   De manera similar a la respuesta anterior, el mayor consumo de memoria en el stack se produce en los mismos escenarios. La única diferencia leve radica en el método de implementación.

   El desglose del consumo de memoria en el stack es el siguiente:
   1) Llamada inicial al procedimiento imprimirArbol

2) Si el nodo no es nulo
2.1) $\longrightarrow$ Realizo un push con el valor del nodo.
3) Si subarbol izquierdo no es nulo
3.1) $\longrightarrow$ Realizo una llamada recursiva a la función para la rama izquierda.

Con esta variación, el consumo de memoria en el stack se mantiene prácticamente invariable, siendo distinto únicamente al procesar el nodo en la altura M.
En este caso, dicho nodo no ejecutará el paso 3, por lo tanto, el consumo de memoria en el stack será de $(1 + 2 \cdot M - 1)$ bytes, lo que eequivale a $(2 \cdot M) \cdot 2$ bytes.