

Práctico 8 – Recursión en Intel 8086.

Objetivo

Familiarizarse con el pasaje de parámetros en la arquitectura Intel 8086, en particular en el contexto de rutinas recursivas y con el manejo de diferentes contextos de ejecución.

Notas

- Es recomendable trabajar con una cartilla de instrucciones para realizar el práctico. La cartilla es provista por el cuerpo docente en exámenes.
- Se deben resolver todos los problemas en alto nivel y luego compilarlos a assembler.
- Excepto cuando se indique explícitamente, las variables están almacenadas en el segmento DS.

Preguntas teóricas:

- Explique por qué es importante salvar el contexto en una rutina recursiva en assembler 8086.
- Explique detalladamente qué realizan las instrucciones `call` y `ret`.
- ¿Qué relación hay entre el tamaño de un segmento y el máximo de llamadas recursivas a una función?

Ejercicio 1 ★★ (en OpenFing)

Se dispone de una lista enlazada de caracteres representada mediante un array. Cada elemento del array contiene un carácter y el índice (dentro del array) del siguiente elemento de la lista. La lista vacía se representa mediante un índice con valor negativo. Específicamente, la variable `Lista` se define mediante la siguiente declaración:

```
struct Item {
    char c;
    short siguiente;
}
Item lista[N];
```

La función recursiva `ConstruirCadena` construye una cadena de caracteres representada de acuerdo a la convención del lenguaje C, a partir del contenido de la variable `Lista`:

```
void ConstruirCadena(short indiceLista, char *bufferDestino) {
    if (indiceLista < 0)
        *bufferDestino = '\0';
    else {
        *bufferDestino = lista[indiceLista].c;
        ConstruirCadena(lista[indiceLista].siguiente, bufferDestino + 1);
    }
}
```

Se solicita compilar la función `ConstruirCadena` en assembler Intel 8086.

El parámetro `indiceLista` se pasa en el registro BX y la variable `Lista` está apuntada por DS:SI. El parámetro `bufferDestino` se pasa en la pila como un desplazamiento con respecto al segmento apuntado por ES y la función no retira este parámetro del stack.

Una llamada a la función se realiza mediante los siguientes pasos:

1. Se asigna el valor del parámetro `indiceLista` al registro BX.
2. Se coloca `bufferDestino` (desplazamiento con respecto a ES) en el tope de la pila.
3. Se llama a `ConstruirCadena`.
4. Se retira el parámetro `bufferDestino` de la pila.

Ejercicio 2 ★★★

Dada la secuencia de naturales $S(0), S(1), \dots, S(N)$ definida por la relación:

$$S(N) = S(N-1) + S(N-2). \text{ Siendo } S(0)=0 \text{ y } S(1)=1.$$

- A) Escribir en un lenguaje de alto nivel una función recursiva $F(N)$ tal que $F(N)=S(N)$.
- B) Compilar la rutina en assembler 8086. El argumento se pasa en el registro AX y el valor de la función se devuelve en el registro BX.
- C) Calcular la cantidad de bytes mínima que debe tener el stack para que sea posible la ejecución de la función en el caso $N=5$.

Ejercicio 3 ★★

Se considera la siguiente estructura de árbol binario, donde `hijoIzq` e `hijoDer` son los índices a los subárboles izquierdo y derecho respectivamente para un nodo dado.

```
struct Nodo {
    short dato;
    unsigned char hijoIzq, hijoDer;
};
Nodo arbol[256];
```

El árbol tiene por lo menos un elemento. El valor 0 en `hijoIzq` o `hijoDer` significa que ese nodo no tiene el sucesor correspondiente.

- A) Escribir en alto nivel una función *recursiva* que busca un entero en el árbol y devuelve true si está y false en caso contrario. La función recibe como argumentos el entero a buscar y un índice al árbol o subárbol donde buscar.
- B) Compilar la rutina en assembler 8086 sabiendo que en AX se recibe el entero y en BX el puntero al árbol. El árbol se encuentra cargado en memoria a partir de la posición 0 del ES. El resultado se devuelve en el registro CL (1 es TRUE y 0 es FALSE). Se deben conservar todos los demás registros.

- C) Calcular el tamaño mínimo que debe tener el stack para que la función pueda ser ejecutada en todos los casos, cualquiera sea el tamaño del árbol.

Ejercicio 4 ★★ (en OpenFing)

Se considera un árbol binario cuyo nodo es definido de la siguiente manera (izquierdo y derecho son punteros a los dos subárboles del nodo):

```
struct Nodo {
    Nodo* izquierdo;
    Nodo* derecho;
    short numero;
};
```

El árbol no tiene por qué estar balanceado. El valor NULL en cualquiera de los nodos (derecho o izquierdo) significa que el nodo no tiene un sucesor por la correspondiente rama del árbol.

- A) Escribir en un lenguaje de alto nivel una función recursiva que calcula la profundidad del árbol (largo máximo de caminos entre la raíz y un nodo)
- `short profundidad(Nodo* arbol);`
- B) Compilar la rutina en assembler 8086. El programa llamador hace la siguiente invocación:

```
"PUSH segmento árbol"
"PUSH offset árbol"
CALL profundidad
"POP profundidad"
```

El resultado se devuelve en el stack y los argumentos deben retirarse del stack. Se deben conservar todos los registros.

Nota: los punteros en este ejercicio son *far* y se representan en la estructura `Nodo` como 2 palabras (offset y segmento del puntero).

- C) Calcular el tamaño del stack necesario en el peor caso para un árbol con N nodos.

Ejercicio 5 ★★★

Se considera un árbol binario cuyo nodo es definido de la misma manera que en el ejercicio 4.

- A) Escribir en un lenguaje de alto nivel una función recursiva que calcula la cantidad de nodos que tengan exactamente dos hijos.

```
short cantidadBifurcaciones(Nodo* arbol);
```

- B) Compilar la rutina en assembler 8086. El programa llamador hace la siguiente invocación:

```
"PUSH segmento árbol"
"PUSH offset árbol"
CALL cantidadBifurcaciones
```

“POP cantidad”

El resultado se devuelve en el stack y los argumentos deben retirarse del stack.

- C) Calcular el tamaño mínimo de stack para que la función pueda ser ejecutada en todos los casos, cualquiera sea el tamaño del árbol.

Ejercicio 6 ★★★

El problema de las Torres de Hanoi se describe a continuación:

Sean tres torres, donde la primera tiene n discos concéntricos, cada uno de los cuales tiene menor diámetro que el disco que debajo del mismo. La segunda y la tercera torre están inicialmente vacías.

El objetivo es transferir todos los discos a la tercera torre, uno por vez, y en ningún momento un disco puede estar encima de otro que tenga diámetro menor.

El algoritmo que resuelve el problema sería el siguiente:

```
/*
* n: nro de discos total
* i: nro de discos en la torre inicial
* j: nro de discos en la torre objetivo
*/

void torres (short n, short i, short j) {
    short k;

    if (n==1) {
        moverDisco(i, j); /* Mueve disco de torre i a j */
    } else {
        k= 6-i-j;
        torres(n-1, i, k);
        torres(1, i, j);
        torres(n-1, k, j);
    }
}
```

- A) Compilar la rutina en Assembler 8086, sabiendo que n viene en el stack y que la rutina moverDisco es una rutina ya definida que recibe los parámetros en el stack.
- B) Indicar el tamaño mínimo que tiene que tener el stack para resolver el problema con $n=3$.