Práctico 5

Tema: Assembler 8086. Recursión en 8086. Entrada/salida en 8086.

Objetivo: Familiarizarse con las instrucciones, registros y modos de direccionamiento del microprocesador 8086. Compilar rutinas recursivas manipulando adecuadamente el stack. Realizar accesos a E/S e implementar rutinas de atención a los dispositivos (programada o interrupciones).

Nota:

- Se deben resolver todos los problemas en alto nivel y luego compilarlos a assembler.
- Excepto cuando se indique las variables están almacenadas en el segmento DS.

Ejercicio 1
Implementar una rutina en assembler 8086 que dados: D1 (dirección de memoria), D2 (dirección de memoria) y N, toma N bytes consecutivos comenzando en D1 y los copia a partir de D2. D1 viene dado en el registro AX, D2 en BX y N en CX.
Nota: AX no puede usarse para acceder a memoria.

Ejercicio 2
Un compilador emplea la siguiente manera de traducir la comunicación entre una función y el programa que la invoca:

- Los argumentos de la función (que se puede suponer todos de 16 bits) se envían al stack en el orden en que son declarados en la llamada a la función.
- La subrutina que implementa la función deja los argumentos en el stack y devuelve el resultado en el registro AX, se deben mantener los valores de los demás registros.

Se pide:

(a) Compilar en Assembler 8086 una función que haga el AND de tres palabras de 16 bits (cumpliendo con las reglas anteriores):

```
function AND(x, y, z: integer) return integer;
```

(b) Escribir un programa Assembler 8086 que calcule el AND de los registros AX, BX y CX llamando a la rutina de la parte anterior.

Ejercicio 3
Se considera la siguiente estructura de árbol binario:

```
type nodo =
  record of
    dato: integer;
    hijoIzq,hijoDer: byte;
  end
arbol = array ☯..(MAX_NODOS-1)☸of nodo;
```

donde:

dato es un entero de 16 bits.
hijoIzq e *hijoDer* son índices a los subárboles izquierdo y derecho respectivamente.

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

Se pide:

- 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. Compilar la rutina en assembler 8086 sabiendo que en AX se recibe el entero y en BX el puntero al árbol o subárbol. El árbol esta cargado en memoria como un array de nodos a partir de la posición 0 del Extra-Segment. El resultado se devuelve en el registro CL (1 es TRUE y 0 es FALSE). Se deben conservar todos los demás registros.
- 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 (Ex. Arq. 2 22 de diciembre de 2010)

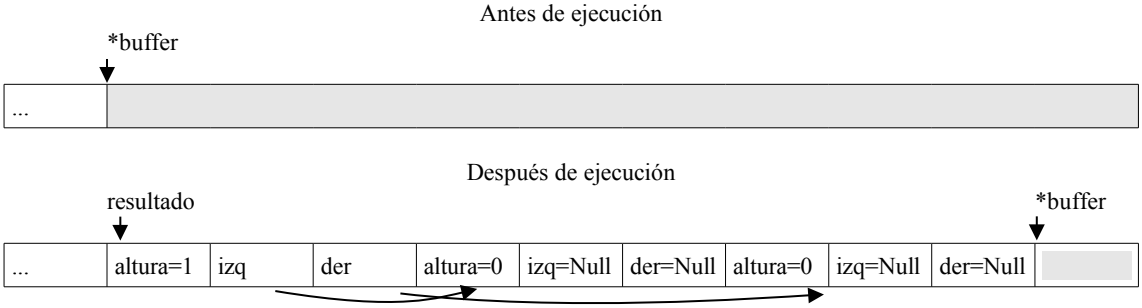
Considere la siguiente estructura de datos de árbol y la declaración de la función `crearArbol`.

```
typedef struct {
  unsigned int altura;
  *nodo izq, der;
} nodo;

nodo *crearArbol (nodo **buffer, unsigned int profundidad)
```

La función `crearArbol` crea un árbol completo y balanceado (todo nodo interno tiene dos hijos y todas las hojas están a la misma profundidad). Recibe por parámetro la profundidad del árbol a crear (un árbol con un solo nodo tiene profundidad cero) y devuelve un puntero a la raíz del árbol creado. El campo `altura` de cada nodo contiene la distancia del nodo a las hojas que descienden de él (`altura` vale cero para una hoja).

El parámetro `buffer` determina el espacio de memoria donde crear los nodos del árbol. Específicamente, `*buffer` es un puntero a un área de memoria donde el llamador espera que se cree el árbol. La función `crearArbol` modifica el valor de `*buffer`, de acuerdo a la cantidad de memoria que haya utilizado, para que apunte a la primera dirección libre después de haber creado el árbol. La siguiente figura ilustra un posible resultado de ejecución para una llamada con el parámetro `profundidad` igual a 1.



Se pide:

A) Escriba en lenguaje de alto nivel la función `crearArbol`. Compile en ensamblador de 8086. Los parámetros y el valor de retorno se pasan en el stack. Se debe conservar el valor de todos los registros. Todos los punteros, incluido el parámetro `buffer`, se representan mediante un desplazamiento Near con respecto a `DS`. El puntero NULL se representa con un cero y se asume que buffer es distinto de NULL.

B) Calcule el mayor valor de `profundidad` para el cual su implementación funciona correctamente.

Ejercicio 5 (Ex. Arq. De Comp. 25 de julio de 1997)

Se considera un edificio con un sistema anti-incendios controlado por un microprocesador 8086 dedicado a este propósito. El sistema está constituido por sensores de humo distribuidos en todo el edificio que comparten una línea de interrupción. Cuando un sensor detecta un determinado nivel de humo, genera una interrupción por nivel, atendida por la rutina HUMO. Cada sensor tiene asociada una luz en el tablero del guardia de seguridad y un difusor de agua. La luz debe encenderse cuando se produce una interrupción, y se debe abrir el difusor de agua si transcurren 30 segundos sin que se oprima un botón de reconocimiento de la alarma. Esta acción debe además apagar la luz en el tablero de alarmas.
Se dispone de un timer de hardware que produce una interrupción cada 200 ms, atendida por la rutina TICK; cuando se oprime un botón de reconocimiento de alarma se produce una interrupción, atendida por la rutina ACK.
Cada grupo sensor-difusor-luz-botón tiene asociado un byte de control B_MANDO_n, donde n es el número de grupo. En total existen 10, alojándose consecutivamente a partir de la dirección de E/S 40H. Cada byte de control tiene la siguiente estructura:
Bit 0: reservado
Bit 1: en 1 si el sensor generó una interrupción, se pone en 0 por hardware al ser leído
Bit 2: se debe escribir en 1 para abrir el difusor de agua
Bits 3: se debe escribir en 1 para encender la luz de alarma, en 0 para apagarla.
Bits 4: se pone en 1 cuando se oprime el botón de reconocimiento, se pone en 0 por hardware al ser leído
Bit 5 al 7: reservados

Se pide:

- Programar en un lenguaje de alto nivel las rutinas necesarias para implementar el sistema.
- Compilar en assembler 8086.

Ejercicio 6 (Ex. Arq. 2 19 de febrero de 2010)

Consideramos un dispositivo dedicado a evaluar expresiones aritméticas en notación prefija. Recordamos que en esta notación los operadores preceden a los operandos, lo cual elimina la necesidad del uso de paréntesis. La siguiente tabla muestra ejemplos de expresiones en notación prefija y la expresión equivalente en la notación infija habitual.

Notación prefija	Notación infija
+ 2 3	2 + 3
+ 2 * 3 4	2 + 3 * 4
* 2 + 3 4	2 * (3 + 4)

El dispositivo en cuestión utiliza un procesador 8086 para leer desde un puerto una secuencia de operadores y números que forman una expresión en notación prefija. Cada vez que se recibe una expresión completa, esta se evalúa y el resultado se escribe en otro puerto. El proceso se repite de forma continuada; la notación prefija permite identificar el final de cada expresión sin necesidad de utilizar delimitadores. Por ejemplo, la secuencia + 2 3 + 2 * 3 4 define dos expresiones aritméticas, + 2 3 y + 2 * 3 4, que generan la salida de dos números, 5 y 14, respectivamente.

Cuando se recibe un operador o un número, se genera la interrupción INT_ENTRADA y el elemento recibido queda disponible en el puerto de un byte de lectura ubicado en la dirección PUERTO_IN. El bit más significativo del byte recibido indica si se trata de un número o un operador. Si el bit más significativo es cero, los siete bits menos significativos se interpretan como un entero sin signo. Si por el contrario el bit más significativo es uno, los siete bits menos significativos indican un operador de acuerdo a la siguiente correspondencia: 0 – suma, 1 – resta, 2 – multiplicación, 3 – división.
Cada vez que se completa una expresión, el resultado se evalúa en precisión de entero con signo de 16 bits y se emite por el puerto de escritura de una palabra ubicado en la dirección PUERTO_OUT.

1) Escriba en un lenguaje de alto nivel y compile en 8086 todas las rutinas necesarias para implementar el sistema propuesto. Asuma que no hay divisiones por cero y todos los resultados son representables en 16 bits con signo.

2) Si las expresiones están acotadas a un máximo de N componentes (incluyendo números y operadores), cuánta memoria de datos y stack se necesita para el correcto funcionamiento de su implementación.

Sugerencia: una función para evaluar expresiones prefijas podría tener el siguiente pseudocódigo.

```
int evaluar()
  c = siguiente elemento de la expresión;
  if (c es un número)
    return c;
  else
    case c of
      suma : return evaluar() + evaluar();
      resta: return evaluar() - evaluar();
      mult : return evaluar() * evaluar();
      div  : return evaluar() / evaluar();
```

Ejercicios complementarios

Complementario 1
Se desea implementar una rutina que determine si un string es parte de un texto, devolviendo la posición de comienzo de la primera ocurrencia o un -1 si no se encuentra. El largo de ambos es variable siendo el del string menor o igual que el del texto. Los dos contienen el carácter especial NULL que marca su fin.
Se pide:

(a) Escribir una función en lenguaje de alto nivel que realice la tarea descrita. Texto y string se reciben como parámetros.
(b) Compilar la rutina anterior en assembler 8086. DS:SI apunta a Texto y DS:DI a String. Devolver el resultado en AX.
(c) Discutir si podría aumentarse la velocidad del algoritmo en assembler si se conocieran los largos de los strings.

Complementario 2 (compresión)
Se desea compactar un string de manera que se sustituyan n caracteres consecutivos iguales por la secuencia ESC n char; donde ESC es el carácter ESCAPE (1Bh), n es un byte que indica la cantidad de veces que se repite char y char es el carácter repetido. El string termina con el carácter NULL (0h). El ESC no esta en el string.
Se pide:

(a) Implementar una rutina en alto nivel que realice la compactación. La rutina recibe el string a compactar en STRING_ENT y devuelve el string compactado en STRING_SAL.
(b) Compilarla en assembler 8086. Definir y especificar detalladamente como se realiza el pasaje de parámetros.

Complementario 3
Dada una fecha AAMMDD en una cierta dirección DIR, hallar la fecha del día siguiente y almacenarla en la misma dirección. AA, MM, DD ocupan un byte cada una y son enteros sin signo. No tener en cuenta los años bisiestos. Escribir un programa en assembler 8086, considerando que DIR es 0100:0100.

Complementario 4 (Ex. Arq. 21 de marzo de 2002)

Se considera la siguiente estructura para representar un árbol binario en memoria:

```
typedef struct {
      int dato;
      int izq;
      int der;
} nodo;
nodo arbol[MAX_NODOS];
```

La variable global *arbol* es un array de nodos, donde en cada nodo el campo “dato” contiene un número entero (16 bits) y los campos “izq” y “der” son los índices (de 16 bits) de la ubicación, dentro del array, de los hijos izquierdo y derecho respectivamente.
El nodo en el índice cero del array no se utiliza y un valor cero en izq o der indica que el nodo no tiene hijo por esa rama.
Se desea implementar una función que imprima el contenido del árbol utilizando la función auxiliar “*imprimirDato*”. Esta función es dada y no debe implementarse.

Para cada nodo debe imprimirse primero el contenido del subárbol izquierdo, luego el dato del nodo y finalmente el contenido del subárbol derecho.
Los cabezales de las funciones son:

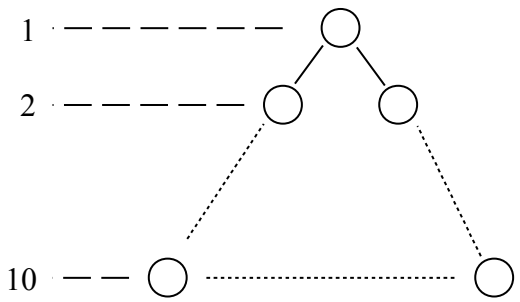
```
void imprimirArbol (int indice_raiz);
void imprimirDato(int dato);
```

Se pide:

(a) Implemente la función *imprimirArbol* en un lenguaje de alto nivel.
Compile la función *imprimirArbol* en ensamblador 8086 sabiendo que:

- La función *imprimirDato* recibe su parámetro en el stack y lo retira. Puede modificar el valor de cualquier registro (es decir no conserva los registros).
- La función *imprimirArbol* debe recibir su parámetro en el stack y retirarlo. No es necesario que conserve los valores de los registros.
- Al momento de la invocación el registro ES apunta al segmento donde se encuentra almacenada la estructura (el elemento 0 del array está al comienzo del segmento)

(b) Si se dispone de N bytes de stack y la función *imprimirDato* consume K bytes de stack incluyendo su parámetro y la dirección de retorno, calcule una cota superior para K de modo que sea posible imprimir el contenido del siguiente árbol completo:



Complementario 5 (Ex. Arq. 23 de febrero de 2003)
Se considera un árbol binario cuyo nodo se define de la siguiente manera:

```
struct nodo {
      long int puntero_izq;
      long int puntero_der;
      int dato;
};
```

donde:

(c) puntero_izq y puntero_der son direcciones segmentadas de 32 bits (16 bits más significativos indican el segmento y los 16 bits menos significativos el desplazamiento), que apuntan a los nodos hijo izquierdo e hijo derecho respectivamente. Están almacenados en memoria siguiendo la regla de Intel (respecto a la ubicación de la parte baja y la parte alta).

(d) dato es un entero de 16 bits

El árbol tiene por lo menos un elemento y no tiene porqué estar balanceado. El valor 0 en cualquiera de los punteros (derecho o izquierdo) significa que el nodo no tiene un sucesor por la correspondiente rama del árbol.

Se pide:

- Escribir en un lenguaje de alto nivel una función recursiva que devuelve la cantidad de nodos con exactamente dos hijos. La función recibe como argumento el puntero al (sub)árbol a recorrer. Compilar la rutina en assembler 8086. El programa llamador hace la siguiente invocación:

```
PUSH puntero_segmento
PUSH puntero_desplazamiento
CALL nodos_dos_hijos
POP resultado
```

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

- Calcular el tamaño máximo del árbol (cantidad de nodos) para que la función pueda ser ejecutada cualquiera sea la topología del mismo, sabiendo que se dispone un segmento completo para el stack.

Complementario 6 (Ex. Arq. De Comp. 23 de diciembre de 1996)
Tiro al blanco es un sencillo juego controlado por un procesador 8086.
Consiste en 8 lámparas que se iluminan en secuencia a intervalos de 0.1 segundos.
Una lámpara encendida representa el blanco y el objetivo del jugador es presionar el interruptor correspondiente mientras la lámpara esté encendida. Si se logra el objetivo, la lámpara indicada debe permanecer encendida. Gana el jugador que logre disparar a los 8 blancos en el menor tiempo posible.
La lámpara e-nésima (0..7) se enciende poniendo en 1el bit e-nésimo del puerto de E/S 11h (de lectura / escritura).
Se dispone de un timer que genera una interrupción cada 10 ms y que es atendido por la rutina Timer.
Al presionar un interruptor se genera una interrupción que es atendida por la rutina Interruptor y en el byte del puerto de E/S 10h (de sólo lectura) se escribe un dígito entre 0 y 7 correspondiente al ÚLTIMO interruptor presionado.
El juego se inicia al escribir 1 en el bit menos significativo del puerto de E/S 12h (de sólo escritura). Al finalizar deberá guardarse en la dirección TIEMPO_TOT el tiempo empleado por el participante en completar su objetivo. Si éste no se logra en un tiempo menor a 900 segundos el juego debe finalizar escribiendo B0B0H en TIEMPO_TOT.

Se pide:

- Implementar en un lenguaje de alto nivel el juego Tiro al blanco descripto.
- Compilarlo en Assembler 8086.