Práctico 7 – Assembler Intel 8086

Objetivo

Familiarizarse con las instrucciones, los registros y los modos de direccionamiento del microprocesador Intel 8086. Resolver ejercicios de programación simples en Intel 8086.

NotaS:

- Es recomendable trabajar con una cartilla de instrucciones para realizar el práctico. La cartilla de instrucciones se entrega a los estudiantes en los exámenes.
- Se deben resolver todos los programas en alto nivel y luego compilarlos a assembler.
- Excepto cuando se indique algo diferente, las variables están almacenadas en el segmento DS.

Preguntas teóricas

(a) Explique cómo se obtiene la siguiente instrucción a ejecutar al comienzo de un ciclo, en la arquitectura 8086.
(b) ¿La arquitectura 8086 es RISC o CISC? Justifique su respuesta.
(c) ¿Qué es una dirección segmentada? ¿Cómo es transformada por el 8086 para generar una dirección física?
(d) ¿Qué cuidado se debe tener si existen segmentos que se solapan en el rango de direcciones que abarcan?
(e) Describa detalladamente qué hacen las instrucciones PUSH y POP. ¿Qué registros están implícitamente relacionados a estas operaciones?
(f) Indique al menos otras dos instrucciones que también hagan uso de registros implícitos.
(g) ¿Qué banderas aritméticas son manejadas por el procesador?

Ejercicio 1 ⋆⋆

Escribir una rutina en assembler 8086 que reciba dos números en los registros AX y BX, que los sume y en caso de que se produzca overflow, deje a partir de la dirección 0x0800:0x0100 el contenido de los registros AX, BX, CX, DX, SI y DI.

Ejercicio 2 ⋆⋆

Implementar una rutina en assembler 8086 que dados D1 (dirección de memoria, dada en el registro AX), D2 (dirección de memoria, dada en el registro BX) y N (dado en el registro CX), toma N bytes consecutivos comenzando en D1 y los copia a partir de D2. Las direcciones de memoria son desplazamientos con respecto al registro ES.

Ejercicio 3 ⋆⋆

Implementar una rutina en assembler 8086 que compare los N bytes a partir de D1 con los N bytes a partir de D2 y devuelva Z=1 si son iguales y Z=0 en caso contrario. D1, D2 y N están en registros. Las direcciones D1 y D2 son desplazamientos con respecto al registro DS.

Ejercicio 4 ⋆⋆

Implementar una función que dados una tabla de bytes y un elemento, indica si el elemento está o no en la tabla. La función debe retornar la posición relativa en que se encuentra el elemento o -1 en caso de que el elemento no se encuentre en la tabla.
Compilar en assembler 8086 la rutina que recibe en AL el elemento a buscar, en BX la dirección de comienzo de la tabla y en DI la dirección final de la tabla (ambas direcciones son desplazamientos con respecto a ES). El valor a retornar se debe almacenar en el registro CX.

Ejercicio 5 ⋆⋆

Implementar una rutina que sume en orden ascendente los elementos de un vector de 1024 palabras, ubicado a partir de la etiqueta VECTOR y retorne en AX la posición del sumando en el cual ocurre CARRY. Si no se produce acarreo al finalizar las sumas se retornará el valor 0. Considere que cuando la rutina es invocada, VECTOR está apuntado por ES:BX.

Ejercicio 6 ⋆⋆

Implementar en assembler 8086 una rutina para eliminar de una cadena (string) todos los caracteres "espacio" (0x20). El string comienza en la dirección ES:BX y el resultado debe almacenarse a partir de la etiqueta ES:DI. Se supondrá que no hay superposición entre el string inicial y el string final. El fin del string estará marcado por el carácter NULL.

Ejercicio 7 ⋆

Indique cómo se distribuye en memoria y cuántos bytes ocupa la variable mi_var, que se define a continuación

```
// Esto es una declaración de tipo, no de variable. No ocupa memoria.
struct mi_tipo{
	char letras[4];
	short enteros[2];
	char letras_MAYUS[2];
};
// Variable mi_var es de tipo mi_tipo
mi_tipo mi_var[3];
```

Ejercicio 8 ⋆⋆⋆

Considere la tarea de determinar si un string es parte de un texto, retornando la posición de comienzo de la primera ocurrencia o el valor -1 si el string no se encuentra. El largo de ambos es variable, siendo el del string buscado menor o igual que el del texto. El carácter especial NULL marca el final del string buscado y el final del texto

a. Implementar en lenguaje de alto nivel la función short substr(char* texto, char* buscado) que realice la tarea descrita previamente.
b. Compilar la rutina de la parte A en assembler 8086. Considere que DS:SI apunta a texto y DS:DI a buscado, y que el resultado se debe retornar en AX.
c. Discutir si podría aumentarse la velocidad del algoritmo en assembler, en caso de que conocieran los largos de ambos strings.

Ejercicio 9 ⋆⋆⋆

Considérese el valor resultante de sumar todos los elementos de una matriz cuadrada de enteros con signo, que se denominará *peso*. Se desea implementar un programa en Intel 8086 que recorra un arreglo de matrices, cada una de ellas de dimensión 4×4, calcule el peso de cada matriz almacene los resultados en un arreglo (el código de referencia se proporciona a continuación). Considere que una matriz se almacena en memoria por filas, con sus elementos en forma consecutiva.

```
Short pesosMatrices[CANT_MATRICES];
Matriz matrices[CANT_MATRICES];
void sumaMatrices{
	for (short i=0; i<CANT_MATRICES; i++){
		// sumar todos los elementos de una matriz
		pesosMatrices[i] = suma de los elementos de matrices[i];
	}
}
```

Para la implementación del programa se disponen de las siguientes constantes:

- MATRICES: indica el desplazamiento dentro de ES donde comienza el arreglo de matrices.
- PESOS_MATRICES: indica el desplazamiento dentro de DS donde comienza el arreglo que se utilizará para almacenar los pesos de las matrices.
- CANT_MATRICES: indica la cantidad de matrices consideradas para la suma (número de elementos en el arreglo de comienza en MATRICES).

Ejercicio 10 ⋆⋆⋆

Un compilador emplea el siguiente protocolo para traducir la comunicación entre una función y un programa que la invoca:

- Los argumentos de la función (se supone que son 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 coloca 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 calcule el AND de tres palabras de 16 bits de acuerdo con el protocolo descrito, con la siguiente firma: `function AND(x, y, z: integer) return integer;`

b. Escribir un programa en assembler 8086 que calcule el AND de los registros AX, BX y CX invocando a la rutina de la parte anterior.