

Laboratorio de Sistemas Basados en Microprocesadores

Práctica 2: Juego de Instrucciones.

Los datos en comunicaciones y los datos almacenados en dispositivos de memoria en condiciones de entorno adversas pueden ser susceptibles a errores que corrompan la información original resultando en datos erróneos. Como ejemplo, en el sector espacial los efectos de eventos únicos (SEE por sus siglas en inglés) inducidos por iones pesados, protones y neutrones se convierten en una limitación constante en el funcionamiento de los componentes electrónicos y pueden hacer que un bit de un dispositivo de almacenamiento se invierta, pasando el valor de una celda de un '0' lógico a un '1' lógico o viceversa.

Durante este laboratorio, se implementará una función llamada EDAC (Detección y corrección de errores por las siglas en inglés) que agregará bits de paridad a los datos para detectar posibles errores de hasta 2 bits por palabra de datos y corregir automáticamente hasta 1 bit por palabra de datos. El desarrollo de un EDAC se utilizará para aprender a usar subrutinas en ensamblador y combinar una gran parte del conjunto de instrucciones del lenguaje ensamblador del 8086.

En esta práctica implementaréis la codificación para Detección y Corrección de un único error, que permita almacenar con seguridad palabras de datos de 4 bits en la memoria.

¿Cómo funciona EDAC?

EDAC utiliza el **código de Hamming** binario que añade bits de paridad para controlar diferentes bits de una palabra, con superposición para favorecer la detección.

Un bit de paridad indica si el número de unos ('1's) en un grupo de bits es par o impar. Si después de la transmisión o almacenamiento, un bit cambia, entonces al volver a calcular la paridad de ese grupo de bits, el valor de paridad será diferente al indicado en el bit de paridad original y significará que hubo un error.

Ejemplo genérico de comprobación de paridad:

1. Palabra de datos original de 9-bits es = "1 0 1 1 0 0 1 0 1".
2. Incluimos 1 bit de **paridad** sobre todos los bits, para asegurar que el número de '1's lógicos en la resultante palabra de 10 bits (9 bits datos + 1 bit paridad) es siempre par.
 - a. Si el número de '1's en los 9-bits es impar, entonces el bit de paridad valdrá '1'.
 - b. Si el número de '1's en los 9-bits es par, entonces el bit de paridad valdrá '0'.

Número de '1's en "1 0 1 1 0 0 1 0 1" → 5,

5 es impar, entonces el **bit de Paridad** valdrá "**1**" y añadimos dicho bit en este ejemplo al final de la palabra así que ahora la palabra de datos será: "1 0 1 1 0 0 1 0 1 **1**"

Ahora el número de '1's es par → 6

3. Transmitimos la palabra de datos y ocurre un error en el segundo bit ('0' cambia a '1'):
"1 0 1 1 0 0 1 0 1 **1**" → "1 **1** 1 1 0 0 1 0 1 **1**"
4. Calculamos la paridad sobre los 10 bits recibidos:
Número de '1's en "1 **1** 1 1 0 0 1 0 1 **1**" es → 7

5. **7 es impar**, por tanto nos damos cuenta que hubo un error en la transmisión, aunque no sabríamos en cuál de los bits.

En esta práctica implementaremos la funcionalidad para detectar y corregir errores de hasta 1 bit. Para una palabra de datos de 4 bits, se agregarán 3 bits adicionales para la paridad de modo que tendremos palabras de 7 bits divididas en:

- 4 bits de DATOS
- 3 bits de PARIDAD

Los bits de paridad se agregarán en las posiciones potencia de 2. En nuestro caso, los bits de paridad estarán en las posiciones 1, 2 y 4. Todos los demás bits son bits de datos. Cada bit de paridad, como se muestra en la Table 1, cubrirá los siguientes bits de datos:

Table 1 Hamming code parity bits H7,4

	P1	P2	D1	P4	D2	D3	D4
P1	x		X		X		X
P2		x	X			X	X
P4				x	X	X	X

Bit Paridad 1 (P1) cubre los bits de datos 1, 2, y 4

Bit Paridad 2 (P2) cubre los bits de datos 1, 3, y 4

Bit Paridad 4 (P4) cubre los bits de datos 2, 3, y 4

EJERCICIO 1: Programa pract2a.asm (2 pts)

Implemente una subrutina "**printASCII**" que convierta un decimal de hasta 16bits a formato ASCII. La subrutina debe recibir el número a convertir en el registro BX, y retornar en DX:AX la dirección de memoria a partir de la que se encuentra la cadena de caracteres ASCII (En DX el segmento y en AX el desplazamiento). La cadena de caracteres debe estar terminada con '\$', a fin de facilitar la impresión mediante la función 9 de la INT 21h del MS-DOS (ver anexo de la práctica y alumno.asm de la práctica 0).

El programa debe probar la rutina "printASCII" sacando en pantalla un valor numérico definido previamente en memoria.

EJERCICIO 2: Programa pract2b.asm (6 pts)

Implemente un programa en ensamblador que cree una subrutina que recibe una cadena binaria de 4 bits y devuelve la codificación hamming EDAC correspondiente como una cadena binaria de 7 bits. La cadena binaria de entrada de 4 bits será predefinida en memoria, donde cada dígito binario se almacenará en un byte. Para ello será necesario desarrollar 2 funciones:

Función Multiplicación de Matrices y Módulo (4 pts): Para calcular los bits de paridad de forma automática, la palabra de datos de 4 bits se multiplicará como un vector de tamaño (1,4) multiplicado por la Matriz de Generación de tamaño (4,7) que se muestra a continuación:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

El valor de esta Matriz de Generación es constante, y se deberá almacenar en memoria.

Siguiendo con el ejemplo anterior, la palabra de datos "1 0 1 1" se multiplicará como sigue:

$$(1 \ 0 \ 1 \ 1) \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} = (1 \ 0 \ 1 \ 1 \ 2 \ 3 \ 2)$$

La salida son 7 bits donde los últimos 3 dígitos corresponden a los bits de paridad P1, P2 y P4. El resultado final se obtiene después de calcular el módulo 2 del resultado de la multiplicación de la matriz:

$$(1 \ 0 \ 1 \ 1 \ 2 \ 3 \ 2) = (1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0)$$

Función Imprimir Resultados (2 pts): El programa mostrará por pantalla el resultado del cálculo en el siguiente formato después de volver a colocar los bits de paridad en la posición correcta:

Input: "1 0 1 1"

Output: "0 1 1 0 0 1 1"

Computation:

	P1	P2	D1	P4	D2	D3	D4
Word	?	?	1	?	0	1	1
P1	0		1		0		1
P2		1	1			1	1
P4				0	0	1	1

Notas:


- La subrutina de multiplicación de matrices recibirá el número de entrada de 4 bits en los registros DX:BX, se almacena 1 bit por cada byte y devolverá en DX:AX la dirección de memoria de la primera posición donde se almacena la salida de 7 bytes (1 bit almacenado por byte). Se recomienda reservar memoria en el segmento de datos o extra para alojar ambos vectores.
- Para obtener acceso a los números en la Matriz de Generación, el alumno deberá usar el direccionamiento indexado a base con desplazamiento fijo.

EJERCICIO 3: Programa pract2c.asm (2 pts)

Modifique el ejercicio 2 para solicitar al usuario que introduzca números del 0 al 15 por teclado. Este número se usará como entrada para la función de codificación EDAC. Utilice las interrupciones del sistema operativo (ver anexo) para capturar este número.

El código ASCII se traducirá al valor decimal correspondiente. El valor decimal se convertirá a binario utilizando el método simple de la división sucesiva por 2 hasta que el resultado sea 0. Cada dígito binario se almacenará en un byte en memoria como en el ejercicio 1. Se imprimirá un error en la pantalla si el usuario introduce un número mayor que 15 o menor que 0.

Ejemplo convirtiendo el número decimal 11 decimal en su representación binaria “1 0 1 1”:

11/2 = 5	Resto = 1	
5/2 = 2	Resto = 1	
2/2 = 1	Resto = 0	
1/2 = 0 (stops)	Resto = 1	
		Binario → 1 0 1 1

Utilice la rutina desarrollada en el ejercicio 2 para imprimir los resultados. Si el usuario teclea “salir” el programa deberá terminar.

ENTREGA DE LA PÁCTICA:

Un único miembro de la pareja deberá subir a Moodle un fichero zip que contenga los ficheros fuente de los programas y el fichero makefile.

Los ficheros fuente deberán contener en la cabecera los nombres de los autores y el identificador de la pareja.

El código generado deberá estar correctamente tabulado y comentado. La falta de comentarios, o la baja calidad de éstos, será calificada negativamente.

Las fechas límite para la subida del archivo son:

Grupo Martes: 02 de Abril de 2018 a las 23:55h
 Grupo Miércoles: 03 de Abril de 2018 a las 23:55h
 Grupo Viernes: 05 de Abril de 2018 a las 23:55h

Anexo: Funciones del sistema operativo para imprimir texto y finalizar la ejecución.

El sistema operativo DOS facilita la mayor parte de sus servicios a través de la interrupción 21h. Antes de que nuestro programa invoque la llamada éste mediante la instrucción INT 21H, se debe cargar en el registro AH el número de la función solicitada. Además, cada función puede requerir de otros parámetros de entrada cuyos valores deberán almacenarse en otra serie de registros. A continuación, se detallan tres funciones de uso habitual en los programas que se desarrollan en el laboratorio.

Función 2H: Envío de un código ASCII al periférico pantalla

Descripción:	Imprime un único carácter por pantalla.
Parámetros de entrada:	AH = 2h. DL = código ASCII del carácter que se imprimirá en pantalla.
Parámetros de salida:	Ninguno.
Registros afectados:	Ninguno.

Ejemplo:

```
mov ah, 2    ; Número de función = 2
mov dl, 'A'  ; Se desea imprimir la letra A
int 21h      ; Interrupción software al sistema operativo
```

Función 9H: Envío de una cadena de caracteres ASCII al periférico pantalla

Descripción:	Impresión de una cadena de caracteres que termina con el carácter ASCII=\$ por pantalla.
Parámetros de entrada:	AH = 9h. DX = Desplazamiento en memoria desde el origen del segmento de datos donde se ubica en memoria el primer carácter de la cadena.
Parámetros de salida:	Ninguno.
Registros afectados:	Ninguno.

Ejemplo:

```
.DATA
Texto DB "Hello world",13,10,'$' ; string terminado con los caracteres CR,LF y'$'
.CODE
.....
; Si DS es el segmento donde está el texto a imprimir:
mov dx, offset Texto ; DX : offset al inicio del texto a imprimir
mov ah, 9            ; Número de función = 9 (imprimir string)
int 21h              ; Ejecuta el servicio del sistema operativo
```

Función 0AH: Lectura de caracteres introducidos por el periférico teclado

Descripción: Captura los caracteres introducidos por el teclado, los imprime por pantalla a modo de eco local (local echo) y se los transfiere posteriormente a la aplicación. La función finaliza cuando el usuario introduce el ASCII 13 (Retorno de carro). El sistema operativo permitirá la entrada de caracteres siempre que no se supere el máximo permitido definido en uno de los parámetros de entrada. En el caso contrario, estos no se mostrarán por pantalla ni se almacenarán.

Parámetros de entrada: DX = Desplazamiento en memoria desde el origen del segmento de datos donde se ubica el espacio reservado para capturar los caracteres. En el primer byte de dicho espacio reservado se ha de almacenar el número de caracteres que se desea capturar.

Parámetros de salida: En el segundo byte de la zona de memoria apuntada por DX el sistema operativo almacena el número de caracteres almacenados. A partir del tercer byte, el sistema operativo almacena los caracteres capturados.

Registros afectados: Ninguno.

Ejemplo:

```
MOV AH,0AH           ;Función captura de teclado
MOV DX,OFFSET NOMBRE ;Área de memoria reservada = etiqueta NOMBRE
MOV NOMBRE[0],60     ;Lectura de caracteres máxima=60
INT 21H
```

;En NOMBRE[1] el sistema operativo almacenará el número de caracteres registrados.
 ;En NOMBRE[2] hasta NOMBRE[61] el sistema operativo almacenará la cadena de
 ;caracteres leídos.

Función 4CH: Fin de programa

Descripción: Esta función indica al sistema operativo que el programa ha finalizado, retornando un valor de ejecución y devolviendo el control.

Parámetros de entrada: AL = 00 (OK) u otro código de error.

Parámetros de salida: Ninguno.

Registros afectados: Ninguno.

Ejemplo:

```
mov ax, 4C00h ; Fin de programa OK
int 21h       ; Ejecuta el servicio del sistema operativo
```