Organización de computadoras

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Segundo Cuatrimestre de 2017

Recuperatorio Primer Examen Parcial Lic. en Ciencias de la Computación – Ing. en Computación – Ing. en Sistemas de Información		
Apellido y Nombre: (en ese orden)	LU:	Hojas entregadas: (sin enunciado)
Profesor:		
NOTA: *Resolver los ejercicios en hojas separadas. Poner nombre, LU y número en cada hoja.*		

Apague cualquier dispositivo electrónico en su poder y manténgalo guardado. No puede utilizar auriculares, ni calculadora. Lea todo el ejercicio antes de comenzar a desarrollarlo.

Ejercicio 1. Dado el número **decimal** $-393{,}3125$ llevar adelante los siguientes cambios de base:

a) Convertirlo a **octal**, empleando el método de la **multiplicación** tanto para la parte entera como para la parte fraccionaria, expresando el resultado en **complemento a la base**, con 4 dígitos octales para la parte entera y 5 dígitos octales para la parte fraccionaria.

b) Convertirlo a **hexadecimal** utilizando el método de la **división** tanto para la parte entera como para la parte fraccionaria, expresando el resultado en **complemento a la base disminuida**, con 5 digitos hexadecimales tanto para la parte entera como para la parte fraccionaria.

Ejercicio 2. Considerando los números **decimales** $X = 142$ e $Y = 113$, llevar adelante las siguientes operaciones, indicando claramente el resultado obtenido y la existencia o no de *overflow*:

a) Calcular $X + Y$, trabajando en **binario** en **complemento a la base**, con una precisión de nueve dígitos (incluido el signo).

b) Calcular $-X - Y$, trabajando en **binario** en **complemento a la base disminuida**, con una precisión de nueve dígitos (incluido el signo).

c) Calcular $X - Y$, haciendo uso de un hardware que opera en una codificación **BCD Exceso-3** y **complemento a la base disminuida**, con una precisión de cuatro dígitos (incluido el signo), indicando claramente qué operación se está realizando en cada uno de los pasos intermedios.

Ejercicio 3. Considerando el Código Cíclico Redundante (CRC):

a) Construir el mensaje $T(x)$ a transmitir asociado al mensaje de dato $M(x) = 100\,1101\,1001$, empleando el polinomio generador $G(x) = x^4 + x^3 + x^2 + x + 1$.

b) Suponiendo que durante la transmisión el mensaje $T(x)$ es modificado y el receptor recibe un $T'(x)$ tal que $T'(x) = T(x) \oplus E(x)$, donde $E(x) = x^2(x^6 + x^2 + 1)$:

b.1) ¿Cuál es el mensaje $T'(x)$ recibido?

b.2) ¿Cuál es la longitud de la ráfaga de error? ¿Es capaz CRC de detectar dicha ráfaga? Justificar.

b.3) Mostrar cómo opera el mecanismo de detección de errores y cuál es la conclusión que se alcanza.

Ejercicio 4. Considerando el código Hamming mínima distancia 3 (Hamming básico), empleando paridad par y estando la secuencia ordenada de izquierda a derecha:

a) Calcular los bits de código asociados al dato 1110 1111 y armar el codeword correspondiente que integra el dato y los bits calculados. ¿Qué tipos de errores es capaz de corregir el código Hamming 3?

b) Considerando que el receptor recibe el codeword 0011 1000 1011 que contiene los bits de dato y de código C_i. Recalcular los bits de código y determinar cuál es el síndrome. ¿Qué conclusión se alcanza bajo la política d=2, c=0.

c) Determinar cómo trabaja el mecanismo de detección/corrección ante una política $d = 1$, $c = 1$, si el síndrome fuera 0001.

Ejercicio 5. Dada la definición para `TCadena`, implementar en **lenguaje C**:

a) Una función `int es_palindroma(TCadena cad)` que dada una cadena de caracteres `cad`, retorne 1 si `cad` es *palíndroma*, y cero en caso contrario. Una cadena de caracteres se dice *palíndroma*, si se lee de igual forma de izquierda a derecha que de derecha a izquierda. Ejemplo: "anana" y "neuquen", son cadenas *palíndromas*, mientras "parcial" no lo es.

b) Una función `TCadena[] clonar (TCadena[] arr, int long)`, que dado un arreglo de cadenas de caracteres `arr` de longitud `long`, retorne un nuevo arreglo de cadenas de igual longitud que `arr`, pero que contenga la *clonación en profundidad* de aquellas cadenas en `arr` que son *palíndromas*. Para esto, contemplar el uso de la función definida en el inciso anterior, así como un correcto uso de la función `malloc` para reservar memoria a la hora de clonar las cadenas.

Dado un *Arreglo A*, la función `clonar()` retornará un nuevo *Arreglo B*, tal como se indica en la siguiente figura:

```
typedef struct cadena{
   char * string;
   int longitud;
}* TCadena;
```